Массовая
радио·
библиотека

МРБ

Т. Н. Поддубная
И. Л. Фукс

# ИНФОРМАТИКА В ЗАДАЧАХ И УПРАЖНЕНИЯХ

Малое предприятие «Раско»

Основана в 1947 году
Выпуск 1167

Т. Н. Поддубная
И. Л. Фукс

# ИНФОРМАТИКА В ЗАДАЧАХ И УПРАЖНЕНИЯХ

МП «Раско»
Томск 1992 г.

ББК 32.973 – 01

П 44

УДК 519.682



**Поддубная Т.Н., Фукс И.Л.**

Информатика в задачах и упражнениях. – Томск: МГП "РАСКО", 1992, 128 с. : ил. – (Массовая радиобиблиотека. Вып. 1167).

Рассматривается ряд задач, связанных с моделированием на ЭВМ реальнах объектов и организацией различных способов обработки числовой и символьной информации. Решение задач доведено до уровня готовых алгоритмов и программ и сопровождается разнообразными упражнениями и заданиями. Подбор задач и характер изложения направлены на иллюстрацию основных понятий и методов информатики.

Для подготовленных радиолюбителей; может быть полезна преподавателям информатики, школьникам и студентам, изучающим этот предмет.

П $\frac{1404000000-041}{046(01)-92}$ 32 – 90 ББК 32.973–01

Рецензент доктор техн. наук, профессор В.А. Кочегуров

ISBN 5-256-00554-5

**АНДРЕЮ ПЕТРОВИЧУ ЕРШОВУ**

**посвящаем**

# ВВЕДЕНИЕ

*На первый взгляд кажется, что регламентация творческого мышления любыми правилами или принципами скорее препятствует, чем помогает этому процессу, но на практике это вовсе не так. Дисциплина мышления мобилизует вдохновение, а не подавляет его.*

***Г.Глэгг***

В одном из своих выступлений Валентин Катаев сказал, что наше общество очень нуждается в подлинно интеллигентных работниках. Содержание интеллигентности со временем меняется, и потому совершенно справедлива мысль о том, что интеллигентность завтрашнего дня немыслима без умения использовать новейшие информационные технологии во всех сферах нашей деятельности, включая и "заповедные зоны" человеческого разума и чувств. Все меньше остается сторон жизни человека и общества, где новинки информатики не приводили бы к ощутимой оптимизации процессов труда и творчества.

Естественно предположить, что большинство выпускников современной школы будет иметь дело с новыми информационными технологиями. "Непонимание научной основы такой технологии не оставляет никаких надежд на то, что человек будет творчески относиться к своей работе, не говоря ужу о том, что такое "информационное невежество" не позволит правильно ориентироваться в соцкультурных явлениях современного мира" [4].

Позитивной характеристикой настоящего времени является тот факт, что информатика перестала быть своего рода экзотикой на ниве народного образования. Споры о необходимости либо о ненужности ее изучения сменила деловая, продуктивная дискуссия содержании нового учебного предмета, методике его преподавания, создании необходимых педагогических средств.

Авторы книги убеждены, что информатика как учебный предмет имеет свое специфическое предназначение в процессе формирования будущего интеллигентного работника. Она значительно больше, чем другие учебные предметы, нацелена на воспитание дисциплинированного мышления. Эта дисциплинированность носит в основном системный характер и потому ни в коей мере не препятствует, а, напротив, способствует развитию творческих компонентов деятельности учащихся.

Умение использовать компьютер для решения практических задач – таким должен быть конечный результат изучения курса информатики в средней школе. Это умение должно основываться на глубоком понимании сути процессов информационного моделирования реальных объектов с помощью компьютера. "Компьютеры не вырабатывают энергию и не варят сталь. Они имеют дело с информацией, знаниями. Поэтому они универсальны" [35]. При внимательном взгляде цепочка "объект – модель – алгоритм – программа – результат" прослеживается практически во всех видах деятельности, связанных с информатикой. Авторы данной книги поддерживают точку зрения, что моделирование становится сейчас незаменимым инструментом нового мышления, "интеллектуальным ядром" информатики. Именно эта точка зрения определила содержание книги – ряд реальных, практически интересных для школьного возраста задач как база для изучения основных приемов информационного моделирования с применением компьютера.

В связи с таким подходом главной целью являлась задача обсуждения и разработки возможных алгоритмов, моделирующих выбранные объекты. Предполагается, что читатель книги знаком с основными понятиями информатики и основными конструкциями хотя бы одного алгоритмического языка. Но надо отметить, что для записи разрабатываемых алгоритмов в книге используется не конкретный алгоритмический язык, а удобочитаемый псевдокод, максимально приближенный к алгоритмическому языку Паскаль и, следовательно, к алгоритмической нотации А.П. Ершова. Выбранный псевдокод будет понятен и тому, кто мало знаком с этой областью знаний.

Авторы стремились прежде всего выпукло отразить структуру алгоритма – модели объекта – и четко определить путь разработки алгоритма – метод последовательной детализации. Именно он способствует формированию у учащегося навыков системного мышления – умения видеть объект моделирования в существенных чертах, как бы "с высоты птичьего полета", абстрагируясь на первых этапах его разработки от частностей и иногда важных нюансов, постепенно опускаясь (а философы говорят – поднимаясь) до их детализации, возможно, и очень сложной.

Сформулированный подход позволяет поставить и достичь определенную педагогическую цель – сформировать у учащегося установку на решение сложной задачи и тем самым заставить работать на уровне его максимальных интеллектуальных возможностей. Опыт авторов (в различных аудиториях: учащихся средних школ, студентов, слушателей курсов повышения квалификации) подтверждает, что в этом случае многочисленные и громоздкие задачи усвоения определенных разделов предметного знания переносятся на уровень естественного (незаметного и прочного) усвоения в процессе решения основной большой задачи. Например, без труда усваиваются достаточно сложные моменты, связанные с выбором различных циклических конструкций в зависимости от вида и характера модели, и способы их правильного оформления. То же самое можно сказать и об организации алгоритмов в виде процедур и функций.

Н. Вирт, автор широко распространенного языка Паскаль, в одной из своих работ [7] сказал: "Создается впечатление, что можно построить

целый курс программирования, выбирая примеры только из задач сортировки". Эта мысль тем более будет справедливой, если выбрать в качестве примеров ряд интересных задач по построению информационных моделей реальных объектов. Достаточно подобрать их таким образом, чтобы они отражали все основные концепции, правила, понятия, законы предмета информатики.

Авторы книги не ставили перед собой цель решить подобную задачу в целом. Но думается, что при желании и соответствующей методической обработке представленные в книге задачи могут способствовать освещению многих разделов названного предмета (основные типы данных, основные действия с ними, способы разработки и записи алгоритмов на конкретных алгоритмических языках).

Работа с арифметическими данными, важность предварительного математического анализа выбранной модели объекта наглядно представлены в задачах "Мухи, слоны и числа" и "Кросснамбер". Разнообразные алгоритмы обработки символьной информации читатель найдет в задачах, связанных с шифровкой текстов ("Тарабарская грамота" и "Компьютер для Вильяма Леграна"). В задаче "Расписание уроков" достаточно подробно представлен логический тип данных и операции с данными этого типа. В задачах "Угадай слово" и "Арифметика для малышей" читатель познакомится с составлением диалоговых программ, обрабатывающих как символьную, так и численную информацию. Широкий набор заданий и упражнений по составлению алгоритмов для обработки информации, заданной в виде массивов (таблиц) данных, представлен в задачах "Автостоянка" и "Хит-парад". Наконец, задачи "Спортлото" и "Маленькая записная книжка" дают возможность на интересном материале познакомиться с более сложными структурами данных – множествами и записями – и способами работы с ними.

Следует также отметить, что внесение в представленные алгоритмы и программы элементов графики, цвета, звука сделает их еще более интересными и будет способствовать достижению поставленной цели – смещению дидактических акцентов в обучении на задачи высокого уровня сложности.

В специальном приложении в книге представлены широко используемые авторами в учебном процессе листы опорных сигналов, являющиеся одним из методических приемов организационно-методической системы В.Ф. Шаталова. Опыт работы с ними при изучении курса информатики показал, что наибольший эффект дает их применение для организации учебных занятий, связанных с обобщением изучаемого материала, его теоретическим освоением. Здесь сказывается специфика информатики как предмета с высоким процентом элемента системности в его содержании. Краткие пояснения к методике работы с листами читатель найдет в указанном приложении, а также в [39, 47].

Нам остается кратко описать используемый в книге псевдокод для записи разрабатываемых алгоритмов. Прежде чем сделать это, следует подчеркнуть, что нашей целью не было соблюдение строгих синтаксических правил какого-то определенного языка. Максимальная понятность записи алгоритма на любом этапе его разработки – вот подход, взятый за основу. Но все же необходимо сказать, что авторы являются

сторонниками использования для обучения основам информатики (в той части, что связана с программированием) алгоритмического языка Паскаль. Почти для всех задач в книге приведены образцы Паскаль-программ для разработанных алгоритмов, а в целом ряде случаев сама разработка идет сразу в ориентации на этот язык (например, в задачах "Маленькая записная книжка" и "Спортлото").

Для записи разрабатываемых алгоритмов используется следующая общая форма:

алг ИМЯ алгоритма большими латинскими буквами

Описания переменных, используемых в алгоритме.

начало алгоритма

Предложения алгоритма, написанные в произвольной форме. Если в одной строке их несколько, они разделяются знаком ;

Можно писать на русском языке.

Оператор присваивания рекомендуется писать, как в языке Паскаль, с использованием знака :=.

Оператор проверки условия записывается в двух видах

– полном:

**проверка:** условие

ДА: действия при "ДА" (в произвольной форме)
НЕТ: действия при "НЕТ" (в произвольной форме)

**конец проверки**

– сокращенном:

**проверка:** условие

ДА: действия при "ДА" (в произвольной форме)

**конец проверки**

Операторы цикла записываются тоже в двух видах

– первый, в котором переход к следующему значению I (параметра цикла) происходит автоматически:

**цикл** по I от нач. знач. до кон. знач.

Действия тела цикла (в произвольной форме).

**конец цикла** по I

Параметр цикла может, естественно, иметь другое имя. Значение шага изменения параметра равно 1.

– второй:

**цикл пока** условие

Действия тела цикла в произвольной форме.
Переход к следующему значению условия

**конец цикла пока**

конец алгоритма ИМЯ.

Эта форма характерна для случая основных, а не вспомогательных алгоритмов. В связи с этим следует сделать два замечания.

1. В алгоритме присутствует раздел описания величин (данных), используемых для обработки и получения результата. В отличие от алгоритмической нотации А.П. Ершова мы не выделяем в записи алгоритма аргументы и результаты, и потому в разрабатываемых алгоритмах должны быть, при необходимости, конструкции ввода исходных данных (ВВОД) и вывода результата (ВЫВОД, ПЕЧАТЬ).

2. Оформление вспомогательных алгоритмов производится в соответствии с алгоритмической нотацией А.П. Ерошова.

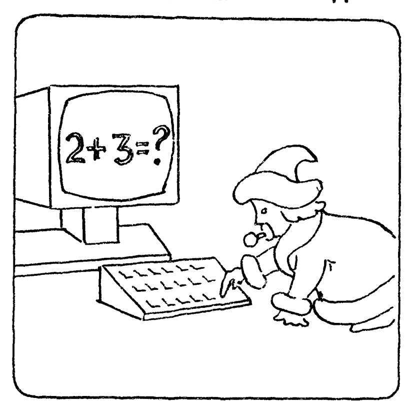
Мы предлагаем читателю относиться к представленному способу оформления алгоритмов в духе времени – плюралистически, помня, что "первейшим достоинством алгоритма является потенциальная компактность рассуждений, на которых может основываться наше проникновение в его сущность" [15], а также то, что "нахождение глубинной простоты в запутанном клубке сущностей – это и есть творчество в программировании" [28].

Свои отзывы, замечания, пожелания по содержанию книги просим направлять по адресу: 634055, Томск, а/я 2211.

*Если вы можете решить задачу, это — упражнение; в противном случае это — проблема.*

***Р. Беллман***

# АРИФМЕТИКА ДЛЯ МАЛЫШЕЙ




*Уча других, также учишься.*

***Н.В.Гоголь***

В вашей школе работает или скоро начнет работать компьютерный дисплейный класс. И вы стремитесь попасть туда: кто поучиться программировать, кто поиграть. Но это старшие ребята. А как же быть малышам ? Им тоже хочется посмотреть новую, такую интересную "думающую" технику и позаниматься на ней. Давайте составим для них программу, одновременно и интересную, и полезную. Дети любят играть – значит, в программе должны быть элементы игры и юмора. Дети должны учиться – пусть программа им в этом поможет.

Поставим себе **ЗАДАЧУ** – написать программу, которая проверяет знания по арифметике у школьников младших классов. Такая контролирующая программа будет и большой помощницей учительнице младших классов, ведь не нужно будет проверять тетради, ЭВМ сама может выставить оценку. Обычно при решении таких задач сначала создается сценарий контрольного урока. Это еще не алгоритм, но как бы первые подходы к нему. Объяснить значение сценария очень просто: мы должны знать, на какие действия настраивать машину, а то, как будут реализованы эти действия, отобразится в алгоритме.

Допустим, мы приняли следующий план сценария:

1) приветствие ЭВМ: знакомство с учеником;

2) краткое объяснение возможностей программы;

3) предложение ученику выбрать одно из четырех арифметических действий, в котором он хочет себя проверить;

4) предложение ученику выбрать сложность примеров (числа до 10, до 1000 и т. п.);

5) печать нескольких примеров с ожиданием ввода ответа учеником;
6) выставление оценки.

## РАЗРАБОТКА АЛГОРИТМА

Общая схема алгоритма должна соответствовать плану сценария:

```
алг ARIFM
    описания
начало алгоритма
   ВЫВОД: приветствие, приглашение к знакомству
   ВВОД: имя или фамилия ученика
   ВЫВОД: что умеет делать программа ARIFM
   ВЫВОД: приглашение к выбору действия
   задание действия
   ВЫВОД: приглашение к выбору сложности примеров
   задание сложности
   цикл по I от 1 до (количество примеров)
     I-й пример
     ответ к I-му примеру
   конец цикла по I
   ВЫВОД: оценка, прощание
конец алгоритма ARIFM.
```

По наличию действий ВВОД и ВЫВОД отмечаем, что программа будет содержать обширный диалог, который должен быть дружественным, незатянутым, но в то же время достаточно подробным и содержательным. Договоримся сразу, что на этапе разработки алгоритма обсуждать фразы, которые будет печатать ЭВМ, мы не станем, укажем лишь их смысл. В конечном варианте программы ARIFM используется одна из форм диалога, вам может понравиться другая форма. Не будем в этом себя ограничивать.

Приступаем к ДЕТАЛИЗАЦИИ отдельных действий алгоритма.

"Задание действия". Здесь должна быть представлена возможность ввода информации (достаточно одного символа, например, первой буквы названия действия: С – сложение, Д – деление и т.п.), по которой ЭВМ затем будет создавать примеры на разные арифметические действия. Таким образом, в алгоритме должна быть символьная переменная, назовем ее D (действие).

"Задание сложности". Принцип реализации этого действия такой же, как и предыдущего, но для разнообразия можно уровни сложности обозначить не буквами, а цифрами, т.е. понадобится целочисленная переменная (назовем ее С) для хранения этой информации.

Переходя к детализации циклической конструкции, отметим, что величина "количество примеров" либо задается заранее, либо вводится в ходе выполнения программы – это решает автор программы совместно с автором сценария.

Мы подошли к действию "I-й пример". Нужно подумать, как напечатать пример заданной сложности на заданное действие, чтобы числа, входящие

в этот пример, выбирались случайным образом. Самое интересное в разработке этого действия то, что оно допускает множество реализаций, непохожих друг на друга. Мы проработаем лишь одно оригинальное направление [49].

Сначала поговорим о числах, входящих в пример. Получить их можно с помощью специальных функций – датчиков случайных чисел. Это функция RANDOM (X) в Турбопаскале (версия языка Паскаль для персональных ЭВМ) или RND (X) в Бейсике. Отличаются они друг от друга тем, что RANDOM (X) генерирует сулчайное целое число из интервала (0,32767), а RND (X) – вещественное число из интервала (0,1). Это означает, что какой бы функцией мы ни воспользовались, нужно как-то изменить интервал получаемых случайных чисел.

Общее правило работы с этими функциями следующее. Первое значение параметра X должно быть задано либо программистом (но тогда при каждом новом выполнении программы последовательности будут повторяться), либо пользователем (тогда программист должен предусмотреть возможность ввода такого значения и известить об этом пользователя в диалоге). Для всех последующих вызовов функции значение аргумента формируется автоматически.

Далее – о нужном интервале значений. Для RANDOM (X) можно воспользоваться операцией MOD – остаток от деления. Результат MOD лежит в промежутке от 0 до (делитель – 1). Например, в самых простых примерах числа не должны быть больше 10. Очередное число вычисляется в операторе присваивания

CHIS : = RANDOM (X) MOD 11.

В Бейсике преобразования несколько сложнее, так как результат RND (X) – вещественное число. Поэтому сначала его нужно увеличить до допустимой величины ("растянуть" интервал путем умножения случайного числа на 11), а потом преобразовать к целому типу. Преобразование к целому типу выполняет функция INT (X), которая отсекает без округления дробную часть операнда X и оставляет целую часть в качестве результата, т.е. INT (4.9) = 4, INT (0.1) = 0, INT (–2.5) = –2 и т.д. Поэтому для получения случайного числа из интервала (0,10) можно выполнить, например, следующее действие:

CHIS : = INT(RND (X) * 11)

Стало ясно, что в действии "I-й пример" для получения чисел из разных интервалов в программе на Бейсике можно применить выражение INT (RND (X) * A), где A связано со значением C – показателем сложности примера. Вычисление этого выражения удобно оформить в виде функции с параметром A, пусть она называется FNP (A). В нашей задаче значение коэффициента A можно представить C-й степенью 10:

число из первого десятка −> C=1 −> A=10,

число из первой сотни −> C=2 −> A=100 и т. д.

Зная C, заданное учеником при работе с программой можно вычислить нужный коэффициент A. Есть и другой путь: создадим заранее целочисленный массив S, например, из 4-х элементов для 4-х уровней слож-

ности и занесем в него значения 10, 100, 1000, 10000. Тогда нужное значение коэффициента А будет равно S (C).

Назовем два случайных числа, нужных для записи примера, именами Z1 и Z2. Запишем детализацию действия "I-й пример":

```
начало
получить  Z1, Z2
проверка: пример на сложение?
  ДА:  вычислить R=Z1+Z2 –результат  сложения
       запомнить знак операции "+"
  НЕТ: проверка: пример на вычитание?
         ДА:  вычислить R=Z1 – результат  вычитания
              исключить отрицательный результат  – Z1=Z1+Z2
              запомнить знак операции "–"
         НЕТ: проверка: пример на умножение?
                ДА:  вычислить R=Z1★Z2 – результат умножения
                     запомнить знак операции  "★"
                НЕТ: вычислить R=Z1 – результат  деления
                     исключить деление на 0
                     исключить дробный ответ Z1=Z1★Z2
                     запомнить знак операции ":"
              конец проверки умножения
       конец проверки вычитания
  конец проверки сложения
  напечатать Z1, знак операции, Z2, знак "="
  конец алгоритма "I-пример".
```

*УПРАЖНЕНИЕ* 1. Пояснить правило формирования примеров на вычитание и деление.

После того, как ЭВМ выведет очередной пример, она должна получить ответ ученика (значение переменной UCH) и сравнить его с правильным ответом R. Для оценки работы ученика нужно подсчитывать количество правильно решенных примеров в счетчике PR. Из перечисленных действий состоит "ответ к I-му примеру".

Более глубокой детализации алгоритма мы проводить не будем. Ниже приводится программа ARIFM, написанная на языке Бейсик.

```
10  DIM S$ (4)
20  DEF FNP (A) = INT (RND★ A)
30  CLS : KEY OFF
40  S (I) = 10
50  FOR I = 2  TO 4
60  S (I) = S (I – 1) ★10
70  NEXT I
80  LOCATE 1,25
90  PRINT "Контролирующая программа"
100 LOCATE 3,15 : PRINT "АРИФМЕТИКА  ДЛЯ  МАЛЫШЕЙ"
110 LOCATE 10,15 : PRINT "ЗДРАВСТВУЙ, ДРУЖОК!"
```

```
120 PRINT "ДАВАЙ, ПОЗНАКОМИМСЯ. МЕНЯ ЗОВУТ ЭВМ, А ТЕБЯ КАК?
130 INPUT IM$
140 PRINT"СКАЖИ МНЕ, ПОЖАЛУЙСТА, ДЕНЬ ТВОЕГО РОЖДЕНИЯ."
150 INPUT DR
160 RANDOMIZE DR
170 PRINT "Я ХОЧУ ПОМОЧЬ ТЕБЕ, "; IM$;", ПРОВЕРИТЬ ЗНАНИЯ ПО
    АРИФМЕТИКЕ."
180 PR=0
190 PRINT "ПОДУМАЙ, КАКИЕ ПРИМЕРЫ ТЕБЕ ХОЧЕТСЯ ПОРЕШАТЬ:"
200 PRINT "НА Сложение, Вычитание, Умножение или Деление?"
210 PRINT "СКАЖИ МНЕ ОБ ЭТОМ, НАБРАВ ПЕРВУЮ БУКВУ ВЫБРАННОГО
    ДЕЙСТВИЯ,"
220 D$=INPUT$(1)
230 IF (D$="С") OR (D$="В") OR (D$="У") OR (D$="Д") GOTO 270
240 PRINT "ЧТО-ТО Я ТЕБЯ НЕ ПОНЯЛА. ПОВТОРИ ЕЩЕ РАЗОК."
250 PRINT "НЕ ЗАБУДЬ, ЧТО Я РАЗБИРАЮСЬ ТОЛЬКО В ЗАГЛАВНЫХ РУССКИХ
    БУКВАХ."
260 GOTO 220
270 CLS : LOCATE 10,1
280 PRINT "А ТЕПЕРЬ ДОГОВОРИМСЯ О СЛОЖНОСТИ ПРИМЕРОВ."
290 PRINT "ЕСЛИ ТЕБЕ     НУЖНЫ ЧИСЛА ИЗ ПЕРВОГО ДЕСЯТКА - НАБЕРИ
    ЦИФРУ 1,"
300 PRINT "                  ДЛЯ ЧИСЕЛ ИЗ ПЕРВОЙ СОТНИ - НАБЕРИ ЦИФРУ 2,"
310 PRINT "                           ДЛЯ ЧИСЕЛ ДО ТЫСЯЧИ - НАБЕРИ ЦИФРУ 3,"
320 PRINT "И НАКОНЕЦ, ДЛЯ ЧИСЕЛ БОЛЬШЕ ТЫСЯЧИ - НАБЕРИ ЦИФРУ 4."
330 INPUT C
340 IF (C=1) OR (C=2) OR (C=3) OR (C=4) GOTO 370
350 PRINT "ТЫ ОШИБСЯ, ТАКОЙ ЦИФРЫ Я НЕ ЗНАЮ. ВНИМАТЕЛЬНО ПОВТОРИ
    НАБОР."
360 GOTO 330
370 PRINT "ВОТ И ХОРОШО! СОБЕРИСЬ С МЫСЛЯМИ, Я НАЧИНАЮ."
380 FOR J=1 TO 1000:NEXT J
390 CLS
400 FOR I=1 TO 10
410 Z1=FNP(S(C)) : Z2=FNP(S(C))
420 IF D$="С" THEN GOTO 460
430 IF D$="В" THEN GOTO 470
440 IF D$="У" THEN GOTO 480
450 GOTO 490
460 ZN$="+" : R=Z1+Z2 : GOTO 520
470 ZN$="-" : R=Z1 : Z1=Z1+Z2 : GOTO 520
480 ZN$="*" : R=Z1 * Z2 : GOTO 520
490 ZN$=":" : R=Z1
500 IF Z2=0 THEN Z2=1
510 Z1=Z1 * Z2
```

```
520 GOSUB 630
530 NEXT I
540 PRINT "ПОСМОТРИ НА СВОИ РЕЗУЛЬТАТЫ:"
550 PRINT "ПРАВИЛЬНЫХ ОТВЕТОВ - "; PR
560 PRINT : PRINT
570 PRINT "ЕСЛИ ХОЧЕШЬ ЕЩЕ ПОСЧИТАТЬ - НАЖМИ 1, ";
580 PRINT "А ЕСЛИ УЖЕ УСТАЛ - ЛЮБУЮ ДРУГУЮ ЦИФРУ."
590 INPUT E
600 IF E=1 THEN GOTO 180
610 PRINT "ДО СВИДАНИЯ, "; IM$; ", ЖДУ ТЕБЯ В СЛЕДУЮЩИЙ РАЗ."
620 END
630 PRINT Z1; " "; ZN$; " "; Z2; "=";
640 INPUT UCH
650 IF UCH=R THEN PR=PR+1
660 RETURN
```

*УПРАЖНЕНИЯ*

2. Добавить в программу ARIFM ввод количества предлагаемых для решения примеров.
3. Видоизменить программу ARIFM так, чтобы она выполнялась заданное число раз.
4. Пояснить смысл оператора RANDOMIZE в программе ARIFM.

**Задания**

А. В каждой программе, где есть операторы ввода, нужно предусматривать проверку вводимых данных, так как человек, работающий с ЭВМ, может ошибиться. Такую последовательность проверок можно назвать фильтром. Найти фильтры в программе ARIFM и объяснить их работу.

Б. Примеры, генерируемые программой ARIFM, не всегда строго соответствуют указанному уровню сложности. Например, для чисел из первого десятка: Z1 =5, Z2 = 5 результат умножения R = 25 – уже не из первого десятка. Придумать такой способ получения примеров, когда и исходные числа, и результат из одного интервала.

# ХИТ-ПАРАД

*— Я не могу этому поверить. Что скажет история?*

*— История, сэр, налжет, как всегда.*

**Б. Шоу**

Как ТАСС проводит хит-парад?

Цитируем "Учительскую газету" от 12.08.89: "Раз в месяц корреспонденты ТАСС в более чем 100 городах нашей страны обращаются в специализированные магазины по продаже грампластинок с вопросом, какие из них в этом месяце пользовались наибольшим спросом. Получив ответ, они передают информацию по каналам ТАСС в центральный аппарат агенства в Москве. Здесь сведения поступают в компьютер, который рассчитывает популярность дисков-гигантов. Так рождается десятка популярности пластинок... Одновременно в ТАСС приходит ежемесячно несколько десятков тысяч писем, в которых любители музыки называют любимых исполнителей и группы. Информация этих писем тоже вводится в компьютер, который подводит итоги популярности среди исполнителей, групп и песен."

Данная задача достаточно объемна и сложна, и потому мы начнем с ее максимальных упрощений, сохраняющих, тем не менее, структуру алгоритмов, обрабатывающих информацию подобного рода. Рассмотрим для начала такой вариант постановки **задачи**.

Имеется постоянная таблица наиболее популярных (в последний год) исполнителей, в которой каждому исполнителю соответствует номер. Например:

| | | |
|---|---|---|
| 1. Ж. Белоусов | 2. В. Добрынин | 3. В. Кузьмин |
| 4. В. Леонтьев | 5. Д. Маликов | 6. М. Муромов |
| 7. В. Пресняков | 8. А. Пугачева | 9. С. Ротару |
| 10. А. Серов | и т. д. ... | 20. В. Цой |

Эта таблица есть у всех корреспондентов ТАСС во всех городах и в компьютерном центре в Москве. Информацию о популярности исполнителей корреспонденты передают в Москву в виде 20 целых чисел, представляющих собой количества купленных пластинок, в порядке определяемом этой таблицей. Например,

В компьютерном центре подобная информация должна быть добавлена к той, что уже хранится в памяти компьютера от других городов. После получения всей информации от всех городов компьютер должен определить 10 наиболее популярных исполнителей по суммарному числу купленных пластинок с их песнями. Таким образом, для этой (очень простой) постановки задачи исходные данные могут быть представлены следующим образом.

Информация в центре – МАССИВ 20 ЦЕЛЫХ чисел, номера которых упорядочены в соответствии с исходной таблицей:

цел таб CENTR [1..20].

Начальные значения элементов этого массива – нулевые.

Информация, поступающая из отдельных городов, – МАССИВЫ аналогичного типа:

цел таб GOROD [1..20].

Таких массивов столько, сколько городов, где есть корреспонденты ТАСС. Предположим, для определенности задачи, что их – 100.

РАЗРАБОТКА АЛГОРИТМА

Самый первый вариант алгоритма очевиден.

```
алг HIT
  цел таб CENTR [1..20]
  цел таб GOROD [1..20]
  цел J, I
  начало алгоритма
    цикл по I от 1 до 100 для всех городов
      ввести информацию об I-м городе
      добавить ее в массив CENTR
    конец цикла
    выделить десятку лучших исполнителей
  конец алгоритма HIT.
```

ДЕТАЛИЗАЦИЯ предложений алгоритма

Из первой записи ясно, что основная конструкция алгоритма – циклическая. Для ее организации использован счетчик с именем I. В этом цикле происходит добавление информации в массив CENTR, следовательно, надо позаботиться о том, чтобы в качестве исходной в него была занесена "нулевая" информация. Таким образом, перед основным циклом появляется дополнительный цикл "обнуления".

```
цикл по I от 1 до 20
   CENTR [ I ] := 0
конец цикла обнуления
```

Ввод информации от I-го города – тоже циклическая конструкция, так как надо ввести 20 целых чисел. Этот цикл организуем с помощью счетчика с именем J.

```
цикл по J от 1 до 20
   ввести значение GOROD[J]
конец цикла по J.
```

Детализируем предложение "Добавить информацию в массив "CENTR". Операция достаточно проста: нужно произвести соответственное сложение значений элементов массивов CENTR и GOROD друг с другом, т.е. мы опять имеем дело с циклической констркуцией. Для ее организации можно использовать счетчик с именем J, примененный в предыдущем цикле для ввода информации о городе, так как он уже свободен.

```
цикл по J от 1 до 20
   CENTR [J]:=CENTR [J]+GOROD [J]
конец цикла по J.
```

Для детализации осталось последнее предложение: "Выделить десятку лучших исполнителей". Можно поступить следующим образом: расположить в порядке убывания элементы массива CENTR, фиксируя в отдельном массиве (назовем его TEN), состоящем из 20 элементов, их старые номера, 10 первых и будут номерами лучших исполнителей в соответствии с заданной таблицей.

До упорядочения:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| массив TEN | 1 | 2 | 3 | 4 | 5 | ... | 19 | 20 |
| массив CENTR | 20 | 265 | 37 | 82 | 400 | ... | 800 | 99 |

После упорядочения:

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| массив TEN | 19 | 5 | 2 | 1 | 20 | ... | 4 | 3 |
| массив CENTR | 800 | 400 | 265 | 120 | 99 | ... | 82 | 37 |

В качестве алгоритма упорядочения (сортировки) можно взять любой, например, сортировку "выбором". Ее идея состоит в следующем: в массиве чисел, которые должны быть упорядочены, отыскивается максимальный элемент, затем он меняется местами с первым элементом и исключается из дальнейшего рассмотрения в алгоритме. Оставшаяся часть массива подвергается той же процедуре. В результате (N – 1) шагов подобного вида, где N – начальное число элементов в массиве, он будет упорядочен. Отметим, что возникающие здесь такие базовые для программирования задачи, как поиск максимальных элементов в числовой последовательности и упорядочение числовых последовательностей, подкрепляются интересом, вызванным исходной большой задачей написания алгоритма хит-парада.

В данном случае важным является следующий момент: всякий раз, устанавливая найденный максимальный элемент на соответствующее место в массиве CENTR, необходимо такую же установку производить и с его старым порядковым номером в исходном неупорядоченном массиве, т.е. менять порядок элементов в массиве TEN. В противном случае мы потеряем связь между номером исполнителя и количеством проданных его пластинок. Получается, что массив TEN изначально должен быть заполнен порядковыми номерами исполнителей. Это можно сделать

совместно с обнулением массива CENTR, теперь эти действия нужно назвать циклом задания исходных значений:

```
цикл по I от 1 до 20
   CENTR [I]:=0
   TEN [I]:= I
конец цикла по I.
```

Ниже приведена запись алгоритма упорядочения по количеству купленных пластинок с запоминанием порядкового номера исполнителя:

```
цикл по I от 1 до N-1
   MAX:=CENTR [I]; NOM:=I;
   цикл по J от I+1 до N
     проверка: CENTR [J] >MAX
       ДА: MAX:=CENTR [J]
           NOM:=J
     конец проверки
   конец цикла по J
   (* максимальное значение - на месте I *)
   CENTR [NOM]:=CENTR [I]; CENTR [I]:=MAX
   (* изменение последовательности старых номеров *)
   (* исполнителей в массиве TEN *)
   C:=TEN [I]; TEN [I]:=TEN [NOM]; TEN [NOM]:=C
конец цикла по I.
```

**Задание А.** Объясните, почему в приведенных циклах стоят именно такие граничные значения параметров цикла?

Детализация закончена, и можно привести весь алгоритм для данной постановки задачи целиком.

```
алг HIT
   цел таб CENTR [1..20]
   цел таб GOROD [1..20]
   цел таб TEN [1..20]
   цел I, J, NOM, C, MAX
     начало алгоритма
     для I от 1 до 20
       CENTR [I]:=0
       TEN [I]:=I
     конец цикла по I
     для I от 1 до 100
       для J от 1 до 20
          ввод информации от I-го города
       конец цикла по J
       для J от 1 до 20
          CENTR [J]:=CENTR [J]+GOROD [J]
       конец цикла по J
```

```
        конец цикла по I
        для I от 1 до 19
          MAX:=CENTR [I];  NOM:=I
     для  J от I+1 до 20
       если CENTR [J] > MAX
         то MAX:=CENTR [J]
            NOM:=J
     конец цикла по J
     CENTR[NOM]:=CENTR [I];  CENTR [I]:=MAX
     C:=TEN [I];  TEN [I]:=TEN [NOM];  TEN [NOM]:=C
   конец цикла по I
   для I от 1 до 10
     напечатать I-й элемент массива
   конец цикла по I
 конец алгоритма HIT
```

**Задания**

Б. Объясните назначение и вид выделенных в алгоритме операторов. Почему в них в качестве индекса используется параметр цикла I ?

В. Нельзя ли в теле первого цикла по I сократить до одного число внутренних циклов по J ? Запишите модифицированный фрагмент алгоритма, если это возможно.

Г. Определите, как будет работать написанный алгоритм в случае, если в массиве CENTR есть одинаковые по величине элементы, т.е. несколько исполнителей претендуют на одно место по количеству купленных пластинок.
Модифицируйте алгоритм для этого случая.
Ниже приведена программа этого алгоритма на языке Паскаль.

```
PROGRAM HIT1 (INPUT, OUTPUT);
(* отладочная программа обработки данных хит-парада *)
CONST N=5;
      M=20;
      L=10;
TYPE MAS=ARRAY [1..M] OF INTEGER;
VAR CENTR, GOROD, TEN : MAS;
(* TEN - массив кодов исполнителей *)
(* GOROD - массив пластинок, купленных в определенном городе *)
   I, J, K, NOM, C, MAX : INTEGER;
BEGIN
  FOR I:=1 TO M DO BEGIN CENTR [I]:=0;  TEN [I]:=I END;
  FOR I:=1 TO N DO
    BEGIN WRITELN ('введите информацию',I,'-го города');
          FOR J:=1 TO M DO
            BEGIN WRITELN (J,'-й исполнитель:');
                  READ (GOROD [J]);
                  CENTR [J]:=CENTR [J] + GOROD [J]
            END
```

```
   END;
 WRITELN ('десятка лучших исполнителей');
 (* алгоритм сортировки и формирование массива TEN *)
 I:=1;
 WHILE I<=M-1 DO
   BEGIN
     MAX:=CENTR[I]; NOM:=I;
     FOR J:=I+1 TO M DO
       IF CENTR[J]>MAX
       THEN BEGIN MAX:=CENTR[J]; NOM:=J END;
     CENTR[NOM]:=CENTR[I]; CENTR[I]:=MAX;
     C:=TEN[I]; TEN[I]:=TEN[NOM]; TEN[NOM]:=C;
     I:=I+1;
   END;
   (* конец алгоритма сортировки *)
   (* заключительная печать *)
     FOR I:=1 TO L DO
     WRITELN('на ',I,'-м месте ',TEN[I],'-й исполнитель');
END.
```

*УПРАЖНЕНИЯ*

1. Модифицируйте алгоритм для случая, когда входная информация для обработки имеет вид двумерного массива цел таб GOROD [1..100, 1..20]. Иными словами, она сначала накапливается в компьютерном центре, затем представляется в виде двумерного массива и только после этого обрабатывается.
2. Примените в алгоритме другой способ сортировки.
3. Модифицируйте алгоритм таким образом, чтобы печатался список десяти лучших исполнителей и пяти исполнителей, занявших последние пять мест.
4. Напишите алгоритм для случая, когда корреспонденты присылают в центр информацию в следующем виде:

   цел таб GOROD [1..40],

где все нечетные элементы массива являются кодами исполнителей в соответствии с приведенной выше таблицей. Все четные элементы – количества купленных пластинок этих исполнителей. Например:

12 164 3 120 8 269...19 400.

В этом массиве: 12, 3, 8,..., 19 – коды исполнителей, а 164, 120, 269,..., 400 – количество купленных пластинок соответственно.

Таким образом, если J – нечетное число от 1 до 39, то в элементах GOROD [J] содержится код исполнителя, а в элементах GOROD [J + 1] – число пластинок. Существенным моментом является то, что коды исполнителей в присылаемых корреспонденциях не упорядочены по возрастанию (см. приведенный выше пример).

Прежде чем продолжить список упражнений, следует обсудить вопрос о таблице исполнителей. Пример ее был приведен в самом начале этой задачи. Для первого варианта постановки задачи мы предположили, что в этой таблице 20 кодов наиболее популярных исполнителей года. Именно из них выбирается десятка лучших, и никто другой, по нашему мнению, не может попасть в списки претендентов, присылаемые корреспондентами из разных городов. Реальная ситуация, безусловно, может быть совсем иной – неожиданно на призовые места начинают претендовать исполнители, кодов которых нет в таблице.

Скорее всего, информация, присылаемая корреспондентами, должна иметь следующий вид:

| N | Фамилия исполнителя | Количество пластинок |
|---|---|---|
| 1 | А. Пугачева | 999 |
| 2 | И. Корнелюк | 1129 |

и т. д.

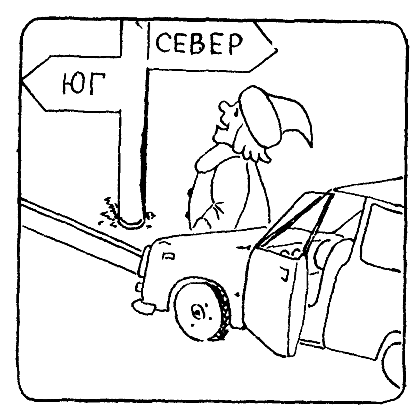
Будем считать, что для корреспондентов число возможных претендентов по-прежнему – 20, т.е. в присылаемых ими списках не содержится более 20 фамилий. Возникает вопрос, насколько объемным сделать массив CENTR в компьютерном центре обработки информации хит-парада? Учитывая фактор моды, увеличим его на порядок. Пусть размерность этого массива для определенности равна 100. Предположим, что массив CENTR в компьютерном центре ТАСС состоит из 100 элементов вида:

| Номер исполнителя | Фамилия исполнителя | Количество пластинок |
|---|---|---|
| NOM | F10 | KOL |

Каждый элемент этого массива содержит в себе три компонента. Два из них (первый и третий) – целого типа, а второй – типа символьный массив. Если вы уже познакомились с задачей "Маленькая записная книжка", то знаете, как представляется информация подобного вида при разработке алгоритмов и как с ней обращаться. Далее предлагается несколько *УПРАЖНЕНИЙ* для этого случая:

5. Модифицируйте алгоритм для предлагаемого типа компонентов массива CENTR при сохранении всех остальных условий первоначальной постановки задачи.
6. Добавьте в алгоритм фрагмент, составляющий таблицу всех исполнителей, фамилии которых поступают в корреспонденциях из городов.
7. Дабавьте в алгоритм фрагмент, по которому включение исполнителей в массив CENTR происходит только в том случае, если число купленных пластинок для него превосходит некоторое заданное число.
8. Придумайте сами несколько упражнений для данной задачи.

# АВТОСТОЯНКА




*Мы, иностранцы, неопытные путешественники. Давно уже, при выезде из нашей Гишпании, мы потеряли компас и потому нечаянно заехали на север.*

***Козьма Прутков,***
*"Любовь и Силин" (драма)*

"Автостоянка содержит одну полосу, на которой может быть размещено до 10 автомашин. Машины въезжают с южного конца стоянки и выезжают с северного. Если автомобиль владельца, пришедшего забрать его, не расположен севернее всех остальных, то все автомобили, стоящие севернее его, удаляются из гаража, затем выезжает его машина, и оставшиеся машины помещаются назад в том же порядке. Если машина покидает гараж, то все машины, расположенные южнее, сдвигаются вперед столько раз, сколько имеется свободных позиций в северной части". Так описана возможная автостоянка в [27].

Поставим **задачу** – построить программную модель этой автостоянки. Прежде всего следует решить вопрос о том, каким образом представлять информацию об автостоянке. Остановимся на следующем: автостоянка – последовательность мест, на каждом из которых может находиться или не находиться автомобиль. По условию задачи таких мест – 10. Иными словами, автостоянка может быть представлена массивом из 10 элементов. Очевидно, что тип элементов бинарен (есть автомобиль, нет автомобиля), поэтому в качестве модели можно взять массив, состоящий из элементов логического типа, т.е. таких, которые могут принимать только одно из двух значений – ИСТИНА или ЛОЖЬ. Более подробно о логическом типе можно прочитать в задаче "Расписание уроков". Сейчас же достаточно сказанной информации о двух возможных значениях. Итак, значению элемента массива ИСТИНА (возможны обозначения ДА, TRUE) соответствует наличие автомобиля, а значению ЛОЖЬ (НЕТ,

FALSE) – его отсутствие.

На рис. 1 приведен пример для логического массива из 5 элементов (все рисунки далее будут сделаны для такого массива). В соответствии с условиями задачи указаны север (N) и юг (S). Состояние автостоянки: на ней находятся 4 автомашины, так как 4 элемента массива имеют значение ИСТИНА, обозначенное буквой T.

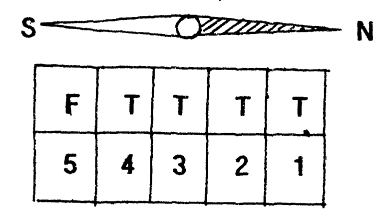



Рис. 1

Очевидно, что для построения модели понадобится еще один такой же массив – дополнительный, на который должны перемещаться автомобили, находящиеся севернее того, который должен выехать. Таким образом, модель гаража – два логических массива, которые опишем следующим образом:

<u>лог</u> <u>таб</u> A [1 .. 10]
<u>лог</u> <u>таб</u> D [1 .. 10]

На основой полосе находятся 4 автомобиля (рис. 2), а дополнительная полоса свободна.

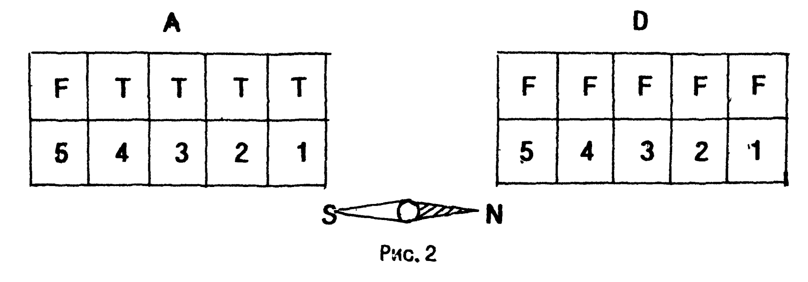



Рис. 2

## РАЗРАБОТКА АЛГОРИТМА

Наша цель – составить алгоритм, моделирующий возможные перемещения в гараже. При выбранных типах данных очевидно, что это есть реализация различных перемещений значений элементов в массивах A и D. На рис. 3 показаны последовательные перемещения автомобилей для случая, когда из гаража должна выехать машина, занимающая третье место в массиве A.

В самом общем виде алгоритм может быть записан так:

<u>алг</u> AVTO
подготовка исходного состояния автостоянки
заполнение автостоянки M (M < 10) автомобилями

выезд автомобиля, занимающего К-е место
**конец** алгоритма AVTO.

A

| 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|
| F | T | T | T | T |
| F | T | F | T | T |
| F | T | F | F | T |
| F | T | F | F | F |
| F | T | F | F | T |
| F | T | F | T | T |
| F | F | T | T | T |

D

| 5 | 4 | 3 | 2 | 1 |
|---|---|---|---|---|
| F | F | F | F | F |
| T | F | F | F | F |
| T | T | F | F | F |
| T | T | F | F | F |
| T | F | F | F | F |
| F | F | F | F | F |
| F | F | F | F | F |

Рис. 3

Приступаем к ДЕТАЛИЗАЦИИ написанных предложений.

1. Подготовка исходного состояния автостоянки.

Предположим, что исходным состоянием является такое, когда на стоянке (ни на основной, ни на дополнительной полосе) нет ни одного автомобиля. Следовательно, всем элементам массивов A и D должно быть присвоено значение ЛОЖЬ. Это может быть сделано с помощью циклической конструкции, которую организуем с помощью счетчика с именем I.

```
цикл по I от 1 до 10
    A[I]:= ЛОЖЬ
    B[I]:= ЛОЖЬ
конец цикла по I.
```

2. Заполнение автостоянки M автомобилями (M <= 10).

Для каждого автомобиля следует сначала определить свободное место, которое должно быть занято им в соответствии с описанными правилами – автомобиль проезжает по полосе до тех пор, пока не доедет до уже стоящей на ней машины. Только самый первый автомобиль должен подъехать к самому выезду из гаража. Эту ситуацию (с первым автомобилем) в случае, когда исходные данные представлены в виде массивов, удобно смоделировать, используя метод "барьера", названный так Н. Виртом и описанный в задаче "Компьютер для Вильяма Леграна".

Увеличим размер массива А, введя вспомогательный нулевой элемент со значением ИСТИНА (барьер). Тогда во всех случаях (и для первого автомобиля тоже) просмотр массива А для обнаружения свободного места должен заканчиваться, как только встретится элемент со значением ИСТИНА (а он встретится теперь обязательно).

Таким образом, второе предложение может быть детализировано в виде следующей циклической конструкции:

```
цикл по J от 1 до M
  I:=10
  цикл пока A[I] не равно значению ИСТИНА
  I:=I-1
  конец цикла со счетчиком I
  A[I+1]:= ИСТИНА
конец цикла со счетчиком J.
```

Во внутреннем цикле, организованном с помощью счетчика I, происходит последовательный проомотр элементов массива А, начиная с десятого (т.е. с севера на юг), до тех пор, пока не встретится элемент со значением ИСТИНА, означающий, что это место уже занято. Таким образом, имитируется движение автомобиля, въехавшего на стоянку, с южного конца к северному.

3. Теперь детализируем следующее предложение – выезд автомобиля, занимающего К-е место на автостоянке (значение К должно быть введено), из гаража. Это предложение в соответствии с описанием задачи может быть детализировано в следующей последовательности:

3.1) перемещение (К – 1) автомобилей на запасную полосу;

3.2) выезд К-го автомобиля из гаража;

3.3) возврат (К – 1) автомобилей с запасной полосы на основную полосу в прежнем порядке;

3.4) перемещение автомобилей, стоящих перед К-м, на одну позицию севернее.

Детализация предложения 3.1.

Отметим, что перемещение автомобилей на запасную полосу всегда происходит при условии, что она перед началом перемещения полностью свободна. Поскольку переместиться должны (К – 1) автомобилей, предложение 3.1 можно оформить как циклическую конструкцию:

```
цикл по I от 1 до K-1
  D[индекс]:=A[I]
  A[I]:=ЛОЖЬ
конец цикла по I.
```

Детализация будет закончена, если мы запишем выражение для индекса, определяющего место в массиве D, которое должен занять автомобиль. Для этого проследим результат перемещений, показанных на рис. 3: 1-й автомобиль из А будет помещен в конце концов на 4-е место в массив D; 2-й из А – на 5-е в D. В общем случае I-й автомобиль из А будет помещен на [N – (K – 1) +]-е место в массиве D, где N – число мест на стоянке. Таким образом, предложение 3.1 приобретает вид:

```
цикл по I от 1 до K-1
   D[N-(K-1)+I]:=A[I]
   A[I]:=ЛОЖЬ
конец цикла по I.
```

3.2. Предложение 3.2 детализируется очень просто – автомобиль выехал, следовательно, место, занимаемое им, освободилось, и соответствующий элемент массива A стал иметь значение ЛОЖЬ.

3.3. Для удобства рассуждений детализируем теперь предложение 3.3, выполняющее обратную операцию – возврат автомобилей с запасной полосы на основную с сохранением порядка.

```
цикл по I от 1 до K-1
   A[I]:=D[N-(K-1)+ ]
   D[N-(K-1)+I]:=ЛОЖЬ
конец цикла по I.
```

3.4. Предложение 3.4 осуществляет сдвиг элементов массива A, имеющих значение ИСТИНА, на одну позицию вправо (к началу массива). Сдвиг начинается с (K + 1)-го элемента. Это действие может быть выполнено с помощью следующей циклической конструкции:

```
цикл по I от K+1 до 10 пока A[I] = ИСТИНА
   A[I+1]:=A[I]
     [I]:=ложь
конец цикла.
```

Для наглядности работы алгоритма в него можно ввести операторы печати состояний массивов A и D. Поскольку эти операторы одинаковы, где бы они ни применялись, их целесообразно оформить в виде процедуры печати, например, так:

```
процедура ПЕЧАТЬ
   напечатать текст "Основная полоса"
   напечатать в строку значения элементов массива A
   пропустить одну строку
   напечатать текст "Дополнительная полоса"
   напечатать в строку значения элементов массива D
   пропустить две строки
конец процедуры ПЕЧАТЬ.
```

Теперь при необходимости в алгоритме достаточно вызвать эту процедуру по имени ПЕЧАТЬ.

Алгоритм целиком, хоть и длинен, но достаточно прост.

```
алг АВТО
  лог таб A[0..10]
  лог таб D[1..10]
  цел I, J, K, M
начало алгоритма
  ВВОД значения M
```

```
для I от 1 до 10
   A[I]:=ЛОЖЬ
   D[I]:=ЛОЖЬ
конец цикла по I
   ПЕЧАТЬ
для J от 1 до M
   I:=10
   пока A[I]=ЛОЖЬ
     I:=I-1
   конец цикла пока по I
   A[I+1]:=ИСТИНА
конец цикла по J
   ПЕЧАТЬ
ВВОД K
для I от 1 до K-1
   D[11-K+I]:=A[I]
   A[I]:= ЛОЖЬ
конец цикла по I
   ПЕЧАТЬ
A[K]:=ЛОЖЬ
для I от 1 до K-1
   A[I]:=D[11-K+I]
   D[11-K+I]:=ЛОЖЬ
конец цикла по I
   ПЕЧАТЬ
для I от K+1 до 10 пока A[I]= ИСТИНА
   A[I+1]:=A[I]
   A[I]:=ЛОЖЬ
   I:=I+1
конец цикла по I
   ПЕЧАТЬ
конец алгоритма AVTO.
```

*УПРАЖНЕНИЯ*

1. Приведенный выше алгоритм индифферентен к номерам автомобилей, и, вообще говоря, в этом случае он бы мог быть проще. Но зато он достаточно хорошо отражает динамику перемещений автомобилей. В качестве первого упражнения предлагаем такое: модифицируйте алгоритм таким образом, чтобы он реагировал на двузначные целые номера автомобилей.
2. Модифицируйте алгоритм таким образом, чтобы он реагировал на символьные номера типа "ТОЕ 73-49".
3. В алгоритме с использованием логических массивов упростите ту часть алгоритма, в которой производится сдвиг элементов массива A, содержащих автомобили, начиная от K + 1, вправо на одну позицию.

В ситуации, представленной на рис. 4, для организации сдвига достаточно третьему элементу присвоить значение ИСТИНА, а седьмому – ЛОЖЬ, не изменяя значений четвертого, пятого и шестого элементов.

4. Добавьте в алгоритм фрагмент, который проверяет, не заполнена ли полностью основная полоса, и выдает соответствующую информацию.
5. Модифицируйте алгоритм таким образом, чтобы выезжающий автомобиль задавался не своим положением на основной полосе, а своим личным номером.

Рис. 4

| F | F | F | T | T | T | T | F | F | F |
|---|---|---|---|---|---|---|---|---|---|
| 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

6. Если позволяет ваш компьютер, напишите программу, графически отображающую динамику перемещений автомобилей.
7. Составьте алгоритм для более удобной автостоянки, на которой выезд не связан с перемещением впереди стоящих автомобилей. Программа, реализующая алгоритм AVTO на языке Паскаль, приведена ниже.

```
PROGRAM AVTO (I,O);
(* ПРОГРАММА, МОДЕЛИРУЮЩАЯ РАБОТУ АВТОСТОЯНКИ С ОСНОВНОЙ *)
(* И ДОПОЛНИТЕЛЬНОЙ ПОЛОСОЙ НА 5 МЕСТ. НОМЕРА АВТОМОБИЛЕЙ *)
(* ЗАДАЮТСЯ ДВУЗНАЧНЫМИ ЧИСЛАМИ.*)
CONST N=5; M=4;
TYPE MAS=ARRAY [0..N] OF INTEGER;
     MAS1=ARRAY [1..N] OF INTEGER;
VAR A:MAS; D:MAS1; L,I,J,K: INTEGER;
PROCEDURE PECH;
  BEGIN
    WRITELN('ОСНОВНАЯ ПОЛОСА');
     FOR L:=N DOWNTO 1 DO WRITE (A[L]:5);
    WRITELN; WRITELN ('ДОПОЛНИТЕЛЬНАЯ ПОЛОСА');
    FOR L:=N DOWNTO 1 DO WRITE (D[L]:5);
    WRITELN; WRITELN;
  END;
BEGIN
  FOR I:=1 TO N DO BEGIN A[I]:=0; D[I]:=0 END;
  A[0]:=100; (*ЗАДАНИЕ БАРЬЕРА*)
  FOR J:=1 TO M DO
    BEGIN
      I:=N;
      WHILE A[I]=0 DO I:=I-1;
      WRITELN ('ВВЕДИТЕ ДВУЗНАЧНЫЙ НОМЕР ПОДЪЕЗЖАЮЩЕГО
               АВТОМОБИЛЯ');
      READ (A[I+1]);
    END;
```

```
WRITELN ('ИСХОДНОЕ СОСТОЯНИЕ СТОЯНКИ'); PECH;
WRITELN ('КАКОЙ ПО СЧЕТУ АВТОМОБИЛЬ ДОЛЖЕН ВЫЕХАТЬ?');
READ (K);
FOR I:= 1 TO K-1 DO BEGIN D [N+1-K+I]:=A[I]; A [I]:=0 END;
WRITELN ('ПЕРЕМЕЩЕНИЕ НА ДОПОЛНИТЕЛЬНУЮ ПОЛОСУ'); PECH;
A [K] :=0;
WRITELN (K,'-й АВТОМОБИЛЬ ВЫЕХАЛ'); PECH;
FOR I := 1 TO K-1 DO DEGIN A [I] :=D[N+1-K+I]; D[N+1-K+I]
      :=0 END;
WRITELN ('ВОЗВРАТ АВТОМОБИЛЕЙ НА ОСНОВНУЮ ПОЛОСУ'); PECH;
I :=K+1;
WHILE A[I] <>0 DO DEGIN A [I-1] :=A[I]; A [I]:=0; I :=I + 1 END;
WRITELN ('ОКОНЧАТЕЛЬНОЕ СОСТОЯНИЕ СТОЯНКИ'); PECH;
END.
```

## ОТГАДАЙ СЛОВО

*И орел не взмахивал крылами,*
*Звезды жались в ужасе к луне,*
*Если точно розовое пламя*
*Слово проплывало в вышине.*

***Н. Гумилев***

В книге Е.Я. Гика "Занимательные математические игры" описано несколько интересных тестовых словесно-математических игр. Одной из них является игра "Отгадай слово", появившаяся на свет в конце 60-х годов одновременно с "быками и коровами". Автор считает, что она значительно богаче и глубже большинства известных игр, в том числе "балды".

Вот как эта игра описана у Е.Я. Гика: ≪Играют двое. Один игрок задумывает слово из 5 букв, а другой должен его отгадать. С этой целью он называет одно за другим слова, состоящие из произвольного числа букв, на каждое из которых партнер в ответ сообщает число, показывающее, сколько раз буквы задуманного слова входят в названное; при этом

каждая буква задуманного слова учитывается столько раз, сколько она содержится в названном. Приведем пример. Пусть воображаемый партнер задумал слово КОЛБА, а мы своим ходом назвали слово ОБОРОНА. Тогда он должен ответить числом 5. В самом деле, буквы К и Л задуманного слова не входят в названное (или иначе – входят 0 раз), буква О входит 3 раза, буквы А и Б – по одному разу. Итого: 0 + 0 + 3 + 1 + 1 = 5.

Называя некоторое слово и получая на него ответ, мы всякий раз делаем определенные выводы относительно задуманного слова. Так, ответ противника 5 на слово ОБОРОНА означает, что задуманное слово, пока неизвестное нам, обязательно содержит букву О (в противном случае максимальный ответ был бы равен 4), а также две буквы из четырех Б, Р, Н, А. Рассмотрим другие возможности. Ответ 0 свидетельствовал бы о том, что в отгадываемом слове нет ни одной из пяти букв, входящих в слово ОБОРОНА; ответ 1 или 2 – что в нем содержится соответственно одна или две буквы из четырех – Б, Р, Н, А и нет буквы О; ответ 3 – что в нем есть О и нет Б, Р, Н, А или , наоборот, есть три из этих четырех букв и нет О; наконец, при ответе 4 делаем вывод, что задуманное слово содержит букву О и одну букву из четырех остальных или все эти четыре буквы вместе, но тогда отсутствует

Извлекая на каждом ходу ту или иную инфомацию о задуманном слове противника, мы делаем следующий ход и т.д., пока не получим ответ "отгадал"≫.

Поставим **задачу** – написать программу для компьютера, которая могла бы выполнять функции партнера, первоначально задумывающего слово. Согласно описанию игры в этой программе необходимо предусмотреть подсчет числа вхождений букв слова, задуманного компьютером, в слово, названное другим игроком (т.е. нами). Очевидным свойством этой программы является то, что она должна быть диалоговой. Мы говорим компьютеру слово, а он отвечает нам целым положительным числом.

Прежде всего обсудим вид и характеристики данных, с которыми должна работать программа. Это слова из пяти букв, иначе говоря, пятисимвольные строки или символьные массивы, состоящие из пяти элементов. В описании игры сказано, что отгадывающий игрок может назвать слова из произвольного числа букв, но мы ограничим себя (для начала) только пятибуквенными словами. Таким образом, наша программа должна обрабатывать символьную информацию. В определениях этой информации в различных языках программирования нет единообразия. Мы будем ориентироваться на определения, характерные для языка Паскаль. В процессе разработки алгоритма воспользуемся следующими описаниями:

лит таб A [1..M] означает одномерный массив (линейную таблицу) с именем A, состоящий из M символов, а

лит таб SLOVA [1..N, 1..M] означает двумерный символьный массив с именем SLOVA, состоящий из N строк и M столбцов. Если, например, N = 6, а M = 5, то такое описание может представлять собой 6 пятибуквенных слов (элементы массива – буквы, а строки в целом могут рассматриваться как слова). Ниже приведен пример значений такого массива:

робот
порох
кашпо
баран
голос
слеза

## РАЗРАБОТКА АЛГОРИТМА

Выделим следующие основные подзадачи алгоритма:

1. "Задумывание" пятибуквенного слова партнером-компьютером.
2. Ведение диалога.
3. Подсчет числа вхождений букв.

Обсудим **первую подзадачу.** Самый простой вариант "задумывания" может быть реализован следующим образом:

а) ввести в компьютер некоторый словарь из пятибуквенных слов, например, 100 или 200 таких слов;

б) запрограммировать датчик случайных чисел, вырабатывающий случайное целое число в требуемом диапазоне. Сначала можно этот фрагмент программы заменить на оператор ввода целого числа с клавиатуры компьютера, поскольку знакомство с конструированием случайных датчиков может значительно затруднить задачу. Но в перспективе написание программ, генерирующих целые случайные числа, становится предметно-обоснованным. Как написать такой датчик для нужного диапазона, можно посмотреть в задаче "Арифметика для малышей".

2. Введение диалога в программе – **вторая подзадача** алгоритма. Общая структура алгоритма вырисовывается уже достаточно ясно.

<u>алг</u> OTGADA
 ВВОД словаря пятибуквенных слов
 "Задумывание" слова из словаря
 "Отгадывание"
<u>конец</u> <u>алгоритма</u> OTGADA.

ДЕТАЛИЗАЦИЯ предложения "отгадывание"
Естественная запись этого предложения будет такой:

**цикл пока** слово, "задуманное" компьютером, НЕ отгадано
 ввод с клавиатуры слова игроком
 **проверка:** слово игрока совпадает со словом компьютера ?
  ДА: слово отгадано (вывод на экран соответствующей информаци
  НЕТ: подсчет числа вхождений букв задуманного слова в слово, назва ное игроком (вывод на экран соответствующей информаци
 **конец проверки**
**конец цикла** пока.

Отметим, что сообщение компьютера (вывод соответствующей информации) должны быть весьма занимательными, располагающими к

общению, содержащими слова одобрения, поддержки, похвалы, и т.д. Компьютер должен быть приятным партнером.

Продолжаем детализацию. Обсудим вопрос об организации **проверки** совпадения слова, "задуманного" компьютером, со словом, названным игроком.

Так как оба слова представляют собой пятисимвольные массивы, эта проверка должна быть реализована путем посимвольного сравнения элементов массивов. Введем для этого счетчик с именем I. Слово, "задуманное" компьютером, определяется элементами K-й строки, где K – созданное число, неизвестное игроку.

I := 1
**цикл пока** I <= 5
  **проверка:** I-я буква компьютерного слова НЕ равна I-й букве слова игрока
    ДА: зафиксировать факт несовпадения слов
      выйти из цикла по I
    НЕТ: увеличить счетчик на 1
  **конец проверки**
**конец цикла** по I.

Существенным моментом здесь является фиксация факта несовпадения или совпадения слов. Это можно сделать, используя **цикл пока.** В случае появления несовпадающих букв присходит выход из цикла, для чего счетчику цикла может быть присвоено значение, заведомо большее граничного, указанного в заголовке. В нашем случае можно взять, например, 10. Проверяя значение счетчика после выхода из цикла, можно сделать нужный вывод: если оно равно 10, то слова не совпадают и в этом случае следует заняться подсчетом числа вхождений букв. В противном случае – слово отгадано и игра закончена.

Таким образом, результат проделанной детализации будет иметь вид:

присвоить счетчику I значение 1
**цикл пока** I меньше или равно 5
  **проверка:** I-я буква компьютерного слова НЕ равна I-й букве слова игрока
    ДА: присвоить счетчику I значение 10
    НЕТ: увеличить значение счетчика I на 1
  **конец проверки**
**конец цикла пока**

**проверка:** значение счетчика I НЕ равно 10
  ДА: слово отгадано (вывод сообщения)
  НЕТ: подсчет числа вхождений
**конец проверки.**

Займемся теперь детализацией предложения "Подсчет числа вхождений". Операция простая. Имеется два пятибуквенных слова. Пусть

первое – компьютерное, а второе – игрока. Например: БАРАН и БАРИН. Очевидно, что если брать подряд все буквы слова БАРАН и смотреть, сколько раз они встречаются в слове БАРИН, то ответ будет 5. Б – 1 раз, первая А – 1 раз, Р – 1 раз, вторая А – 1 раз и Н – 1 раз. А ответ должен быть равен 4. Ясно, что вторую букву А при подсчете числа вхождений учитывать не надо. Поэтому при организации подсчета нужно обязательно проверять, не встречалась ли в задуманном слове проверяемая буква ранее. Учет этого приводит к такой детализации:

числу вхождений присвоить значение 0
**цикл** для всех букв компьютерного слова от 1-й до 5-й
  **проверка:** проверяемая буква НЕ встречалась в "задуманном" слове ранее?
    ДА: проверка на совпадение букв компьютерного слова и слова игрока
    НЕТ: переход к следующей букве "задуманного" слова
  **конец проверки**
**конец цикла** для всех букв.

Из всех написанных выше предложений особо следует остановиться на проверке: встречалась буква ранее или нет. Рассуждаем так: если сейчас анализируется, скажем, I-я буква "задуманного" слова, то надо проверить, нет ли такой буквы среди первых от 1-й до (I – 1)-й, т.е. мы опять должны применить циклическую конструкцию. Оформим ее с помощью счетчика L:

счетчику L присвоить значение 1
**цикл пока** L-я буква НЕ равна I-й <u>И</u> L < I
  увеличить значение счетчика L на 1
**конец цикла пока.**

Если после выхода из этого цикла значение счетчика L будет равно значению I, это и будет означать, что I-й буквы ранее не было и можно сравнивать ее с буквами слова игрока. Обращаем внимание, что используемый цикл – типа <u>пока</u>.

Теперь можно записать разработанный алгоритм целиком.

<u>алг</u> OTGADA
  ввод словаря пятибуквенных слов
  "задумывание" слова компьютером
  <u>пока</u> слово не отгадано
    I: = 1
    <u>пока</u> I < = 5
      <u>если</u> I-я буква компьютерного слова НЕ равна I-й букве слова игрока
        <u>то</u> I : = 10
        <u>иначе</u> I: = I + 1
    <u>конец</u> <u>цикла</u> <u>пока</u>
    <u>если</u> I = 10

```
        то  выход из цикла пока слово НЕ отгадано
              ВЫВОД соответствующей информации
        иначе
          числу вхождений присвоить значение 0
          для  I  от  1  до  5
           все
          L := 1
          пока  L-я буква "задуманного" слова НЕ равна I-й букве
                   этого   же   слова  И   L < I
            L := L + 1
          конец  цикла  пока
          если   L = I
            то
              для  J от  1 до  5
                если  I-я буква "задуманного" слова совпадает с
                         J-й буквой   слова игрока
                  то  увеличить число ее вхождений на 1
                все
              конец  цикла для со счетчиком J
          все
        конец цикла для со счетчиком I
        печать сообщения о количестве вхождений букв
    все
  конец цикла пока   слово не отгадано
 конец  алгоритма  OTGADA.
```

Ниже приведена программа для этого алгоритма, написанная на языке Паскаль.

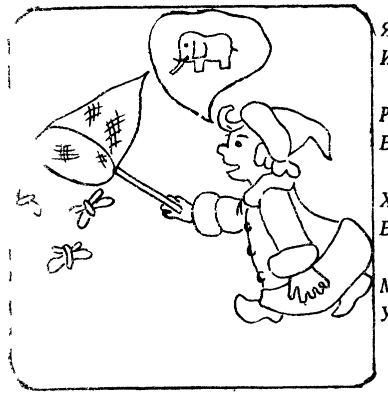
```
PROGRAM  OTGADA  (INP,  OUT);
CONST   N = 10 ;  M = 5 ;
TYPE  MAS1 = ARRAY [1 .. 5] OF   CHAR ;
VAR  A :  MAS1 ;
     SLOVO :  ARRAY [1 .. N, 1 .. M] OF   CHAR ;
     KOL, K, I, J, L :  INTEGER ;
     B : BOOLEAN ;
BEGIN
  WRITELN  ('Введите массив из 10 слов');
  FOR  I := 1  TO  N  DO
    BEGIN
    READLN ;
    FOR  J := 1  TO  M  DO  READ  (SLOVO [ I, J ]) ;
    END ;
    WRITELN  ('Введите любое целое число от 1 до 10') ;
  READ  (K);  WRITELN ;
  WRITELN  ('Спасибо! Я задумала слово из 5 букв');
  WRITELN  ('Отгадай его по следующему правилу:');
```

```
     WRITELN ('Называй любое слово из 5 букв.');
     WRITELN ('Я буду называть в ответ целое число, показывающее');
     WRITELN ('число вхождений разных букв моего слова в твое.');
     WRITELN ('Учти, что я считаю все вхождения.');
     WRITELN ('По этому числу анализируй ситуацию и определяй');
     WRITELN ('возможные буквы задуманного мною слова.');
     WRITELN ('          ЖЕЛАЮ УСПЕХА!');
     B:=TRUE;
     WHILE  B  DO
     BEGIN
     WRITELN ('Называй свое слово '); READLN;
     FOR J:=1 TO M DO READ (A [J]);
     I:=1;
     WHILE  I<=M  DO
        IF SLOVO [K,I] <> A[I]
        THEN I:=M+5 ELSE I:=I+1;
     IF I=M+1 (* СЛОВО ОТГАДАНО *)
     THEN BEGIN
              WRITELN ('Верно, я загадала это слово. МОЛОДЕЦ!');
              B:=FALSE
          END
     ELSE BEGIN        (* Подсчет числа вхождений *)
              KOL:=0;
              FOR I:=1 TO M DO
                BEGIN
                  L:=1;
                  WHILE  (SLOVO [K,L] <> SLOVO [K,I]) AND (L<I) DO
                     L:=L+1;
                  IF L=I
                  THEN
                     FOR J:=1 TO 5 DO
                        IF SLOVO [K,I]=A [J]
                        THEN KOL:=KOL+1;
                END;
          WRITELN ('Буквы задуманного слова входят в твое',KOL,'раз')
        END
   END
END.
```

*УПРАЖНЕНИЯ*

1. Подсчитайте "трудоемкость" этого алгоритма, определив, сколько раз выполняются действия в теле основного цикла.
2. Модифицируйте имеющиеся в алгоритме циклы **пока**, введя логичекую переменную, которая меняет свое значение на противоположное при выполнении соответствующего условия.
3. Введите в алгоритм изменения и дополнения, которые ограничивают возможности игрока, отгадывающего слово. Например, надо угадать слово не более, чем за 10 ходов. Добавьте поощряющий или порицающий игрока диалог.
4. Напишите этот алгоритм на языке Бейсик.

# МУХИ, СЛОНЫ И ЧИСЛА



*Я всматриваюсь в вас, о, числа,*
*И вы мне видитесь одетыми в звери,*
*в их шкурах,*
*Рукой опирающимися на вырванные дубы.*
*Вы даруете единство между змееобразным*
*движением*
*Хребта вселенной и пляской коромысла,*
*Вы позволяете понимать века, как быстрого*
*хохота зубы.*
*Мои сейчас вещеобразно разверзлися зеницы*
*Узнать, что будет Я, когда делимое его —*
*единица.*

***В. Хлебников,** "Числа*

ЭВМ можно научить решать различные задачи-головоломки. Обычно в таких задачах приходится перебирать массу вариантов возможных решений, т.е. многократно повторять некоторый алгоритм проверки данных, шаг за шагом меняя его входные значения до тех пор, пока не найдется ответ. Если по условию предполагается не один ответ, а несколько, то это серьезно усложняет ситуацию при "ручном" решении. Для ЭВМ же не представляет большого труда просмотреть несколько сотен или даже тысяч вариантов. Причем в этом случае ответ уже будет наиболее полным.

Начнем с того, что научим ЭВМ делать из мухи слона. Вернее, нам понадобится не одна муха, а три одинаковых. Предлагается решить ЗАДАЧУ разгадывания числового ребуса [34]:

```
 МУХА
+
 МУХА
+
 МУХА
 ----
 СЛОН
```

Здесь предполагается, что каждой букве соответствует своя цифра, т.е. складываются три однаковых четырехзначных числа, состоящих из разных цифр, а в результате получается тоже четырехзначное целое число, все его цифры не совпадают ни между собой, ни с цифрами исходного числа.

Порассуждаем над решением. Понятно, что организовав прямой перебор всех четырехзначных чисел в поисках МУХИ, можно найти все возможные варианты решения. Алгоритмически такой перебор запишется в виде четырех вложенных циклов, в каждом из которых будет задаваться очередная цифра, т.е. он будет выполняться за 10 шагов. Это означает, что машина $10^4$ = 10000 раз будет проверять, нельзя ли из очередных МУХ получить СЛОНА.



3*

*УПРАЖНЕНИЕ* 1. Какое значение будет иметь переменная SKLAD после выполнения фрагмента алгоритма:

```
SKLAD := 10
для I от 1 до 5
  для J от 3 до 11
    для K от 0 до 7
      SKLAD := SKLAD + 0.5
    конец цикла по K
  конец цикла по J
конец цикла по I.
```

Как сократить количество переборов? Для ответа на этот вопрос подумаем, все ли цифры могут соответствовать буквам слова МУХА. Первая цифра не может быть 0, так как МУХА должна быть четырехзначной. Первая цифра не может быть больше 3, потому что 4* 3 = 12, т.е. СЛОН будет пятизначным. Получается, что вместо буквы М можно попытаться подставить только цифры 1, 2 или 3. Последняя цифра не может быть 0, так как 0* 3 = 0, а это уже СЛОН, тоже оканчивающийся на 0. Но она еще не может быть 5, так как 5* 3 = 15, у СЛОНА вместо Н появится тоже 5. Тогда возможны подстановки вместо А: 1, 2, 3, 4, 6, 7, 8, 9. Исключить из рассмотрения цифру 5 можно только дополнительной проверкой, которая усложнит алгоритм. Договоримся этого пока не делать. О средних двух цифрах сказать что-либо определенное заранее трудно, поэтому оставим диапазон их изменения от 0 до 9.

## РАЗРАБОТКА АЛГОРИТМА

Из предварительных рассуждений заключаем, что в алгоритме будет создано 3* 10* 10* 9 = 2700 чисел – претендентов на звание МУХИ. Каждое из них нужно будет подвергнуть проверке, и тогда в итоге мы получим полный набор возможных решений.

Схема алгоритма будет иметь вид (для наглядности параметры цикла названы буквами слова МУХА):

```
алг МУХА
цикл по М от 1 до 3
  цикл по У от 0 до 9
    проверка: М и У – разные?
      ДА: цикл по Х от 0 до 9
            проверка: М, У и Х – разные?
              ДА: цикл по А от 1 до 9
                    проверка: М, У, Х и А – разные?
                      ДА: получить СЛОНА
                            проверка: СЛОН подходит?
                              ДА: напечатать ответ
                              НЕТ: искать следующую МУХУ
                            конец проверки СЛОНА
                      НЕТ: искать следующую МУХУ
```

```
                    конец проверки М, У, Х, А
                  конец цикла по А
              НЕТ: искать следующую МУХУ
          конец проверки М, У, Х
        конец цикла по Х
    НЕТ: искать следующую МУХУ
  конец проверки М, У
 конец цикла по У
конец цикла по М
конец алгоритма МУХА.
```

Из этой схемы явствует, что любая проверка МУХИ и СЛОНА, закончившаяся негативным ответом, влечет за собой поиск следующего подходящего числа. Но и в том случае, когда ЭВМ нашла искомую пару чисел, она должна продолжить просмотр, так как эта пара может быть не последней возможной. Значит, действия "искать следующую МУХУ" и "следующее значение буквы" – суть одно действие. Это позволит нам не писать в алгоритме при проверке условий ветвь НЕТ, т.е. появляется возможность применить короткую форму условного оператора.

Переходя от алгоритма, записанного в общем виде, к программе, нужно будет решить вопрос о типе циклов, которые мы намереваемся использовать. В нашем распоряжении циклы с заданным числом шагов и с неопределенным числом шагов. По-видимому, рассматриваемая задача реализуется с помощью первого из них. Границы изменения цифр мы определили заранее, шаг их изменения всегда 1, досрочных выходов из цикла не предвидится. Следовательно, останавливаемся на цикле для.

Перепишем алгоритм в виде, близком к программе.

```
для М от 1 до 3 шаг 1
 для У от 0 до 9 шаг 1
  если М < > У
   то для Х от 0 до 9 шаг 1
       если (Х< >М) и (Х< >У)
        то для А от 1 до 9 шаг 1
            если (А< >М) и (А < >У) и (А< > Х)
             то получить СЛОНА
                если СЛОН подходит
                 то напечатать ответ
                все
            все
           конец цикла по А
       все
      конец цикла по Х
  все
 конец цикла по У
конец цикла по М.
```

Дальнейшая детализация алгоритма связана с реализацией действия "получить СЛОНА" и условия "СЛОН подходит". Алгоритм получения СЛОНА удобно оформить в виде вспомогательного. Его основная задача – сложение трех МУХ, выполнить которое можно по-разному.

1. Собрать отдельные цифры – значения переменных М, У, Х и А – в одно число Z (Z = M * 1000 + У*100 + X*10 + A) вычислить Z=Z*3, а затем выделить из значения Z отдельные цифры с помощью операций целочисленного деления и определения остатка от деления, как это рассматривается в задаче "Кросснамбер". Поскольку такое решение в какой-то степени пройденный этап, попробуем найти нечто новое.

2. Выполнить поразрядное сложение значений переменных А, Х, У, М по правилу сложения в столбик. При этом сразу получаются отдельные цифры СЛОНА и возможная цифра переноса в следующий разряд. Для хранения этой цифры нужна специальная переменная. Если после сложения значений М окажется, что перенос не равен 0, это сигнал о том, что СЛОН получился не четырехзначным. Таким образом, значение цифры переноса нужно возвращать в вызывающую программу, чтобы потом оно помогло проверить условие "СЛОН подходит".

Сумма трех цифр находится в диапазоне от 0 до 27. Следовательно, очередную цифру в СЛОНЕ можно получить операцией MOD, делением по модулю, а цифру переноса – DIV, – делением нацело. Например, для суммы, равной 16, очередная цифра – 16 MOD 10 = 6, а цифра переноса – 16 DIV 10 = 1.

*УПРАЖНЕНИЯ*

2. Сформулировать условие задачи, которую можно решить с помощью алгоритма:

```
SI:=0; SJ:=0
для I от 1 до 100
   для J от 10 до 20
      если ((I + J) MOD 5)<>0
         то SJ:=SJ+J
         иначе SI:=SI +I
      все
   конец цикла по J
конец цикла по I.
```

3. Чему будет равно значение переменных SI и SJ после выполнения фрагмента алгоритма:

```
SI:=0; SJ:=0; I:=1
для J от 10 до 20
   если ((I+J) MOD 5)<> 0
      то SJ:=SJ+J
   все
   SI:=SI+I
конец цикла по J.
```

4. Сколько раз проработает цикл во фрагменте программы:

```
I := 0;  Y := 0
пока (I  DIV  100) < 2
    Y := Y + I
    I := I + 10
 конец цикла пока.
```

Получается, что с каждой цифрой МУХИ нужно проделать одинаковые действия, т.е. можно оформить цикл. Но тогда необходимо, чтобы отдельные цифры хранились не как значения простых переменных, а в массиве.

Итак, можно записать алгоритм получения СЛОНА:

```
алг SUMMA (цел  таб MYXA [1:4], SLON [1: 4]; цел PERENOS)
арг MYXA
рез SLON, PERENOS
нач цел SUM, I
    PERENOS := 0
    для I от 4 до 1 шаг −1
       SUM := MYXA [I] * 3 + PERENOS
       SLON [I] := SUM  MOD 10
       PERENOS := SUM DIV 10
    конец цикла по I
конец алгоритма SUMMA.
```

И, наконец, поразмышляем о возможных способах проверки получившегося числа СЛОН. Об одной детали этого действия мы уже говорили. Если алгоритм SUMMA вернет значение параметра PERENOS = 0, то мы получили четырехзначное число. Достаточно узнать, разные в нем цифры или нет и не совпадают ли они с МУХОЙ. Если же PERENOS < > 0, то число не подходит.

Продолжая детализацию алгоритма МУХА и обратившись к условию "СЛОН подходит", нужно сначала проверить значение переменной PERENOS, ибо, как мы уже говорили, PERENOS < > 0 есть сигнал о том, что СЛОН получился не четырехзначный и он, естественно, не подходит. Если же PERENOS = 0, то нужно проверить на несовпадение все цифры СЛОНА и МУХИ. Фактически эта проверка представляет собой самостоятельную оригинальную задачу. Даны два целочисленных массива со значениями элементов, взятыми из интервала [0,9]. Нужно проверить, нет ли среди всех этих элементов совпадающих.

Предлагается следующий путь решения. Организуем целочисленный массив REZ из 10 элементов с номерами от 0 до 9, предварительно обнулим его. Просматриваем массивы MYXA и SLON и прибавляем 1 к значениям тех элементов REZ, номера которых совпадают со значениями элементов MYXA и SLON, т.е. значение элемента одного массива делаем номером элемента другого массива. Если в результате в массиве REZ все элементы будут меньше 2, то это означает, что в массивах MYXA и SLON нет совпадающих значений.

Этот алгоритм удобно оформить в виде функции логического типа,

возвращающей значение ИСТИНА, если все условия выполнены, т.е. в массиве REZ все элементы меньше 2, и ЛОЖЬ – в противном случае. В задаче "Автостоянка" организована работа с логическими переменными. Мы остановились перед необходимостью записать алгоритм, который бы вырабатывал логическое значение. Достаточно полно работа с данными логического типа описывается в задаче "Расписание уроков".

```
лог алг PROVER (цел таб MYXA [1:4], SLON [1:4])
нач цел таб REZ [0:9]
      цел I
      лог PR
      знач := ИСТИНА
      для I от 0 до 9
         REZ [SLON [I]] := REZ [SLON [I]] + 1
         REZ [MYXA [I]] := REZ [MYXA [I]] + 1
      конец цикла по I
      I: = 0
      пока (I <= 9) и знач
         если REZ [I] > 1
            то знач := ЛОЖЬ
            иначе I: = I + 1
         все
      конец цикла пока
конец алгоритма PROVER.
```

В этом алгоритме можно отметить одну особенность, которая довольно часто возникает при работе с логическими данными. Результатом работы функции должно быть одно значение: ИСТИНА или ЛОЖЬ. Еще не выполнив никаких проверок условий, мы оптимистично предполагаем, что результатом будет ИСТИНА (начальное присваивание знач: = ИСТИНА), т.е., что СЛОН подходит. А затем внутри цикла проверяем каждый элемент REZ и в случае выполнения условия REZ [I] > 1 (в СЛОНЕ обнаружилось совпадение цифр) выполняем присваивание знач: = ЛОЖЬ. Поскольку знач входит в условие работы цикла, выполнение последнего завершается, а с ним завершается и работа всего алгоритма, в вызывающий алгоритм возвращается значение ЛОЖЬ. Если же все REZ [I] < = 1, то цикл (и алгоритм) завершится по достижении I = 10, а знач так и останется равным ИСТИНА.

Ниже приводится программа на языке Паскаль, составленная по алгоритму МУХА.

```
PROGRAM TRANS (INPUT, OUTPUT) ;
CONST  S = 9 ;
TYPE  ZIFRA = 0 . . S ;
      MAS = ARRAY [ 1 . . 4] OF  ZIFRA ;
      ZIF = ARRAY [ZIFRA] OF  ZIFRA ;
VAR  MYXA, SLON : MAS ;
      REZ : ZIF ;
```

```
      PERENOS, M, Y, X, A : ZIFRA ;
      I : INTEGER ;
            (* процедура получения СЛОНА *)
PROCEDURE SUMMA (MYXA : MAS ; VAR SLON : MAS ; VAR PERENOS : ZIFRA) ;
   VAR SUM, I : INTEGER ;
   BEGIN PERENOS : = 0 ;
     FOR I : = 4 DOWNTO 1 DO
       BEGIN SUM : = MYXA [I] * 3 + PERENOS ;
         SLON [I] : = SUM MOD (S + 1) ;
         PERENOS : = SUM DIV (S + 1)
       END ;
   END ;
(* функция проверки МУХ и СЛОНА *)
FUNCTION PROVER (MYXA, SLON : MAS) : BOOLEAN ;
   VAR REZ : ZIF ;
       I : INTEGER ;
       PR : BOOLEAN ;
   BEGIN PR : = TRUE ;
     FOR I : = 0 TO S DO REZ [I] : = 0 ;
     FOR I : = 1 TO 4 DO
       BEGIN REZ [SLON [I]] : = REZ [SLON [I]] + 1 ;
             REZ [MYXA [I]] : = REZ [MYXA[I]] + 1
       END ;
     I : = 0 ;
     WHILE (I < = S) AND PR DO
       IF REZ [I] > 1
       THEN PR : = FALSE
       ELSE I : = I + 1 ;
     PROVER : = PR
   END ;
BEGIN
   WRITELN ;
      FOR M : = 1 TO 3 DO
        BEGIN MYXA [1] : = M ;
          FOR Y : = 0 TO S DO
            BEGIN
              IF M < > Y
              THEN
                BEGIN MYXA [2] : = Y ;
                  FOR X : = 0 TO S DO
                    BEGIN
                      IF (X< >M) AND (X< > Y)
                      THEN
                        BEGIN MYXA [3] : = X ;
                 FOR A : = 1 TO S DO
                   BEGIN
                     IF (A<>M) AND (A< >Y) AND (A< >X)
                   THEN
                     BEGIN MYXA [4] : = A ;
```

```
                                    SUMMA (MYXA, SLON, PERENOS) ,
                                    IF  PERENOS = 0
                                    THEN IF PROVER (MYXA, SLON)
                                         THEN  BEGIN
                                          (* печать найденных МУХИ и СЛОНА *)
                                       WRITE (M, Y, X, A, '  ') ,
                                       FOR I := 1 TO 4 DO WRITE (SLON [I])
                                       WRITE ('          ')
                                    END
                                 END
                             END
                          END
                      END
                 END
             END
         END
END.
```

**Задания**

А. Придумать другой алгоритм проверки двух последовательностей с ограниченными значениями на несовпадение этих значений. Сравнить качество своего алгоритма с приведенным выше.

Б. Переписать алгоритм решения ребуса на язык программирования вашей ЭВМ и выполнить его. Сколько МУХ и СЛОНОВ получилось ?

В. Измените полученный алгоритм так, чтобы вместо десятичных МУХ и СЛОНОВ (десятичных чисел) он отыскивал их восьмеричных сородичей. Сложение должно выполняться тоже в восьмеричной системе счисления. Сколько таких пар чисел у вас получилось ?

Тем, кого заинтересовало составление алгоритма разгадывания ребуса, советуем познакомиться с книгой [30]. Две задачи оттуда приводим ниже:

```
 _ШЕСТЬ |ДВА              ПРИМЕР
  ТИС    ТРИ               РИМЕР
  ___                    +  ИМЕР
  _АВТ                        МЕР
   РЬЕ                         ЕР
   ___                          Р
   _ИЕЬ                   ______
    ИЕЬ                   ЗАДАЧА
```

Г. Группа "Видеобалет" объявила о наборе в основной состав. Через все преграды творческих конкурсов прошли 147 участниц. Художественный совет в затруднении: нужно отобрать максимальное число девушек, имеющих одинаковый рост, но не более 50 человек. Составить алгоритм, позволяющий провести такой выбор, учитывая, что самый низкий рост 168, а самый высокий – 175 см. Если по указанным критериям не удается выделить менее 50 человек, то алгоритм должен сообщить причины этого. Например, несколько

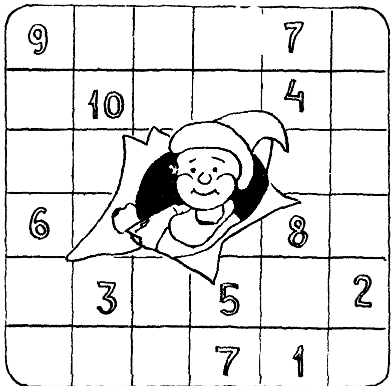
вариантов совокупностей исходных данных (распределение по росту) и желательный ответ алгоритма после обработки этих данных (результат):

| Распределение по росту | Результат |
|---|---|
| а) 168 см – 5 чел.<br>169 – 10<br>170 – 35<br>171 – 20<br>172 – 40<br>173 – 20<br>174 – 15<br>175 см – 2 чел. | Рост 172 см – 40 чел. |
| б) 170 см – 5 чел.<br>172 – 140<br>173 см – 2 чел. | Рост 170 см – 5 чел.<br>дополнительный отбор из 140 чел. ростом 172 см. |
| в) 170 см – 45 чел.<br>171 – 45<br>172 см – 57 чел. | Рост 170 см – 45 чел.<br>рост 171 см – 45 чел.<br>дополнительный отбор из 57 чел. ростом 172 см. |
| г) 172 см – 54 чел.<br>173 – 52<br>175 см – 41 чел. | Рост 175 см – 41 чел.<br>дополнительный отбор из 54 чел. ростом 172 см,<br>дополнительный отбор из 52 чел. ростом 173 см. |

Д Найдите число [33].

В четырехзначном числе все цифры разные и отличны от 0. Если его написать в обратном порядке, получится число, на 8532 меньше первоначального. Найдите это число.

# КРОССНАМБЕР



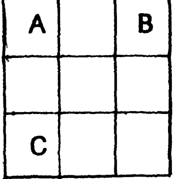

*Целые числа сотворил господь бог, все остальное — дело рук человеческих.*

***Л. Кронекер***

"Что такое кроссворд, знают все. Знают и происхождение этого слова от английских слов "*cross*" ("крест", "пересечение") и "*word*" ("слово").

Словарь определяет кроссворд как "род задачи-головоломки по разгадыванию слов; представляет собой фигуру, разбитую на квадраты, которые нужно заполнить буквами, чтобы по горизонталям и вертикалям получился ряд разгаданных слов". Если учесть, что "number" по-английски "число", то легко понять и значение слова "кросснамбер". Квадраты нужно заполнить цифрами, чтобы по горизонталям и вертикалям получился рад разгаданных чисел.

Например, пусть имеется фигура, состоящая из клеток, некоторые из них помечены буквами А, В, С. Буква отмечает клетку, с которой должна начаться запись числа (будем говорить: число А, число В и т.п.), каждая цифра его занимает клетку. В нашем случае это будет 3 трехзначных числа: А, В и С,




По горизонтали:
$A = X^2$,
$C = (X+Y)*Y+1$

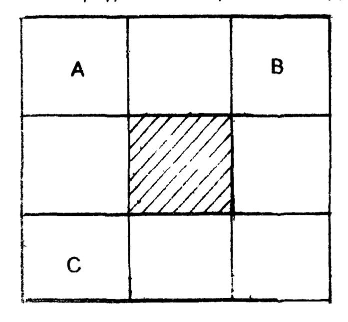
По вертикали:
$B = 2*(X-Y)^2$.

Числа вычисляются по заданным для этого кросснамбера правилам, сформулированным авторами совершенно произвольно. В частности, это могут быть значения функций нескольких переменных (например, X, Y, Z и т.п.). *Требуется найти такую совокупность значений этих переменных, при которой значения функций подходят для заполнения кросснамбера.*

| 2 | 2 | 5 |
|---|---|---|
|  | 0 |  |
| 1 | 0 | 1 |

Так можно заполнить наш кроссгнамбер при $X = 15, \quad Y = 5$

Предлагается решить ЗАДАЧУ




По вертикали.

$A = (W - X)^Z - Z \star W,$

$B = W \star Y \star Z - Z \star W$

По горизонтали.

$A = X^Y,$

$C = Y^{W-Z}.$

Эта задача взята из [32]. По мнению авторов, она является весьма заманчивой, во-первых, потому, что подобные встречаются редко, во-вторых, потому, что "машинный" перебор вариантов позволит узнать, сколько всего существует решений этого кросснамбера.

## РАЗРАБОТКА АЛГОРИТМА

Как и в случае числового ребуса в задаче "Мухи, слоны и числа", нам придется последовательно перебирать значения переменных X, Y, W, Z. Однако если в "Мухах..." сразу были ясны границы изменения переменных M, У, X, A, то в случае кросснамбера сказать что-либо определенное без предварительных размышлений нельзя. Договоримся пока искать решение только среди целых неотрицательных X, Y, W, Z. Введем дополнительные обозначения: AV – A по вертикали; AG – A по горизонтали; ВЕГРX, ВЕГРY, ВЕГРW, ВЕГРZ – верхние границы интервалов изменения X, Y, W, Z соответственно. Алгоритм решения нашей ЗАДАЧИ будет состоять из 4-х вложенных циклов, внутри которых производятся вычисления чисел AV, AG, B, C и проверка возможности заполнения ими кросснамбера.

Договоримся поступать следующим образом. Вычисляем одно из очередных чисел AG, B, C или AV и сразу проверяем, является ли оно положительным трехзначным. Если да, то переходим к определению другого числа. Если же это не так, то меняем параметр цикла, в котором производились вычисления, и вновь подсчитываем одно из AG, B, C или AV. Когда четыре числа AG, B, C, AV получены, проверяем, можно ли ими заполнить кросснамбер. Если да, то печатаем ответ и пытаемся найти другие решения; если нет, продолжаем поиски решения.

Общая схема алгоритма может быть следующей:

```
алг CRNUM
  описание данных
начало алгоритма
цикл по X от 0 до ВЕРГX
  цикл по Y от 0 до ВЕРГY
    вычисление AG
    проверка: AG – трехзначное ?
      ДА: цикл по W от 0 до ВЕРГW
            цикл по Z от 0 до ВЕРГZ
              вычисление B
              проверка: B – трехзначное ?
                ДА: вычисление AV
                    проверка: AV – трехзначное ?
                      ДА: вычисление C
                          проверка: C – трехзначное ?
                            ДА: проверка: можно заполнить кросснамбе
                                  ДА: вывод результата
                                конец проверки заполнения
                          конец проверки C
                    конец проверки AV
              конец проверки B
            конец цикла по Z
          конец цикла по W
    конец проверки AG
    конец цикла по Y
  конец цикла по X
конец алгоритма CRNUM.
```

Первоочередной задачей последующей РАЗРАБОТКИ алгоритма является уточнение границ изменения параметров циклов. Очевидно, искомые величины можно найти, анализируя выражения, по которым вычисляются числа AV, AG, B, C, т.е. мы на время немного отвлекаемся от алгоритма и обращаемся к математическим рассуждениям.

Займемся сначала НИЖНИМИ ГРАНИЦАМИ. Действительно ли они нулевые ? Рассмотрим, например, самое простое выражение $AG = X^Y$. При $Y = 0$ и любом X выражение теряет смысл, значит, нижняя граница Y не может быть нулевой. При $X = 0$ и любом $Y > 0$ – $AG = 0$, т.е. AG не трехзначное. Следовательно, нижняя граница для X не нулевая. Далее, при $X = 1$ и $Y > 0$ – $AG = 1$ – опять не трехзначное число. А вот для $X = 2$ такого простого вывода сделать уже нельзя, поэтому оставим 2 нижней границей для X.

Анализируя аналогичным образом выражения для чисел AV и C, можно определить нижние границы (возможно, не окончательный вариант) для всех переменных X, Y, W, Z. Построим для них таблицу, в которой укажем название числа, чье выражение анализируется, получившуюся нижнюю границу и пояснения по поводу отбрасываемых значений.

| Название числа | Нижняя граница | Причина ограничений |
|---|---|---|
| AV | $X >= 2$ | $AV = 0$ при $X = 0$,<br>$AV = 1$ при $X = 1$ |
| C | $Y >= 2$ | $C = 0$ при $Y = 0$,<br>$C = 1$ при $Y = 1$ |
| AG | $Z >= 2$ | $AG = 1$ при $Z = 0$<br>$AG = -X < 0$ при $Z = 1$ |
| C | $W >= 2$ | C – дробь при $W = 0$,<br>$C = 1$ при $Z = 2$, $W = 1$,<br>C – дробь при $Z > 2$, $W = 1$ |

ВЕРХНИЕ ГРАНИЦЫ для X, Y, W можно подсчитать, воспользовавшись условиями: $AV < 1000$, $B < 1000$, поскольку AV и B – трехзначные. Из $AV < 1000$ следует $X^Y < 1000$. Логарифмируя по основанию $X > 1$, получаем

$$Y * \log_X X < \log_X 1000, \qquad Y < \log_X 1000.$$

Поскольку Y – целое, то последнее неравенство нужно переписать в виде $Y < \lfloor \log_X 1000 \rfloor$, где $\lfloor\ \rfloor$ обозначает ближайшее целое снизу. Например, [3.5]= 3. Наибольшего значения Y достигает при наименьшем $X = 2$:

$Y < \log_2 1000$, т. е. верхняя граница $Y <= 9$.

С другой стороны, из соотношения $X^Y < 1000$ для наименьшего значения $Y = 2$ получаем $X^2 < 1000$; $X < \sqrt{1000}$, т.е. верхняя граница $X <= 31$.

Перепишем выражение для B. $B = W * Y * Z + W = W * (Y * Z + 1)$. При наименьших $X = 2$ и $Y = 2$ $B = W * (2 * 2 + 1) = 5 * W < 1000$, следовательно, верхняя граница $W <= 199$.

Из выражения для C можно определить СООТНОШЕНИЯ МЕЖДУ переменными W и Z. С одной стороны,

$$Y^{W-Z} < 1000.$$

Логарифмируя по $Y > 1$, получим $W - Z < \log_Y 1000$. При наименьшем $Y = 2$ можно найти наибольшее значение разности $W - Z$

$$W - Z < \lfloor \log_2 1000 \rfloor, \qquad W - Z <= 9, \qquad Z >= W - 9.$$

Фиксируя эту нижнюю границу для Z, не нужно забывать о том, что ранее мы вывели соотношение $Z >= 2$. Значит, если окажется, что разность $W - 9 < 2$, то нижней границей для Z должно остаться 2.

С другой стороны,

$Y^{W-Z} >= 100$ в силу трехзначности числа C,
$W - Z >= \log_X 100$.

Подставляя в это неравенство наибольшее значение Y = 9, мы получим наименьшее значение разности W – Z

$W - Z >= \lceil \log_9 100 \rceil$, где $\lceil\ \rceil$ – ближайшее целое сверху ($\lceil 3{,}5 \rceil = 4$);
$W - Z >= 3; \quad Z <= W - 3$.

Поскольку Z > = 2, то из последних неравенств сразу следует, что нижняя граница W > = 5. В алгоритме эта тесная связь между переменными W и Z должна отобразиться во вложенных друг в друга циклах, причем границы внутреннего цикла вычисляются по значениям параметра внешнего цикла. В алгоритме CRNUM внутренний цикл по Z, значит, перед началом его нужно добавить действия по вычислению границ изменения этого параметра.

Итак, выпишем все полученные границы в единую таблицу, чтобы потом при составлении алгоритма легче было ориентироваться в циклах.

| Переменная параметр цикла | Нижняя граница | Верхняя граница |
|---|---|---|
| X | 2 | 31 |
| Y | 2 | 9 |
| W | 5 | 199 |
| Z | 2 или W–9 | W–3 |

Анализируя алгоритм CRNUM, можно дать некоторые рекомендации по поводу его дальнейшей РАЗРАБОТКИ.

1. Проверку числа на трехзначность следует оформить в виде логичекой функции, чтобы не писать четыре раза одно и то же условие. С функцией такого типа мы встречались в задаче "Мухи, слоны и числа".

2. Проверка на возможность заполнения кросснамбера тоже должна быть отдельным алгоритмом, и опять здесь подходит логическая функция, поскольку на наш вопрос возможны только два ответа: ДА или НЕТ.

Запишем функцию проверки целого числа на трехзначность:

```
лог алг TRI (цел A)
нач
  если (A >= 100) и (A < 1000)
    то знач := ИСТИНА
    иначе знач := ЛОЖЬ
  все
конец алгоритма TRI.
```

Прежде чем составить алгоритм для функции, проверяющей возможность заполнения кросснамбера, договоримся об обозначениях. Пусть AV1, AG1, B1, C1 – первые цифры в числах AV, AG, B, C соответственно, а AV3, AG3, B3, C3 – третьи, последние их цифры.

Для выделения отдельных цифр числа нам понадобятся специальные операции над целочисленными данными – MOD и DIV. Результат операции MOD – остаток от деления двух целых чисел, например, 873 MOD 8 равно 1. Очевидно, если в качестве делителя взять основание системы счисления 10, то, выполнив действие 873 MOD 10, мы получим младшую цифру числа – 3. А до остальных цифр нам поможет добраться операция DIV – деление без остатка. Запись 873 DIV 10 фактически означает подсчет количества десятков в числе 873. В полученном результате младшая цифра та, к выделению которой мы стремились, и теперь уже знаем, что для этого нужно сделать 87 MOD 10, т.е. следующая цифра 7, а старшая 8 получится из 87 DIV 10. Так, последовательное чередование MOD и DIV позволяет разложить число на отдельные цифры. В алгоритме эти действия обычно записываются в цикле типа пока.

**Задания**

А. Составить алгоритм, в котором к записи числа N добавляется 1 справа и слева, т.е. из числа 924 получается 19241.

Б. Составить алгоритм, в котором в записи числа N меняются местами крайние цифры, т.е. из 451 получается 154.

В. Составить алгоритм, в котором подсчитывается количество цифр в записи заданного числа N и определяется первая (старшая) цифра записи этого числа.

Г. Составить алгоритм, в котором проверяется, входит ли заданная цифра K в запись числа $N^2$.

Д. Составить алгоритм, в котором порядок цифр в числе M меняется на обратный:

а) с записью результата в другое место памяти;

б) с записью результата на то же место памяти.

Теперь можно записать алгоритм функции, проверяющей возможность заполнения кросснамбера.

```
лог алг CROSS (цел AG, AV, B, C)
нач цел AG1, AG3, AV1, AV3, B1, B3, C1, C3
    знач := ЛОЖЬ
    AG1 := AG DIV 100
    AV1 := AV DIV 100
    если AG1 = AV1
      то AG3 := AG MOD 10
         B1 := B DIV 100
         если AG3 = B1
           то B3 := B MOD 10
              C3 := C MOD 10
              если B3 = C3
                то C1 := C DIV 100
```

```
            AV3:=AV MOD 10
            если C1=AV3
              то знач:=ИСТИНА
            все
         все
      все
   все
конец алгоритма CROSS.
```

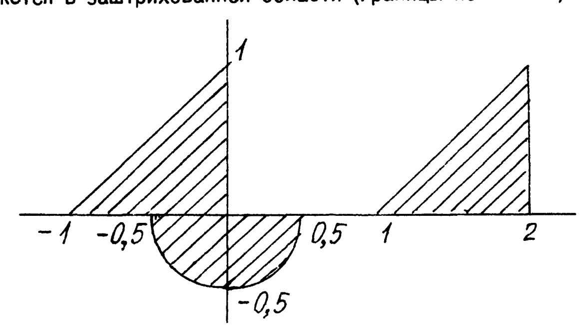
Этот алгоритм, в отличие от функции PROVER, записанной в задаче "Мухи, слоны и числа", можно назвать функцией-пессимистом. Сначала мы предполагаем, что решение не найдено (знач: = ЛОЖЬ до сравнения цифр), а потом делаем попытки доказать обратное. Поскольку все условные операторы вложены друг в друга, т.е. образуют цепочку если – то – иначе, то до последнего условия мы доберемся только при истинности трех предыдущих.

*УПРАЖНЕНИЯ*

1. Какие переменные изменятся в результате работы алгоритма:

```
L:=ИСТИНА; X:=7; Y:=1
если (X>=2) и (X<=5)
  то X:=10
  иначе если (Y>=2) и (Y<=5)
          то Y:=10
          иначе если ((X<=2) и (Y<=2)) или L
                  то Z:=10
                  иначе W:=10
                все
        все
все.
```

2. Записать фрагмент программы, при выполнении которого переменная P получит значение 5, если некоторая точка с координатами (X, Y) окажется в заштрихованной области (границы исключить).




При вычислении AV, C и AG нам понадобится операция возведения

в степень. В некоторых языках программирования, например, в Паскале, таковой нет. Поэтому сразу запишем это действие в виде вспомогательного алгоритма, взятого из [24].

```
цел алг STEP (цел A, N)
нач цел REZ, POK, R
      REZ := 1;  R := A;  POK := N
    пока POK > 0
      если (POK MOD 2) <> 0
        то POK := POK - 1
           REZ := REZ * R
      все
      если POK > 0
        то POK := POK DIV 2
           R := R * R
      все
    конец пока
    знач := REZ
конец алгоритма STEP.
```

**Задание** E. Разобраться в структуре алгоритма STEP и дать его словесное описание. Насколько быстрее по сравнению с последовательным умножением работает этот алгоритм?

С учетом выбранных вспомогательных алгоритмов можно записать программу заполнения кроссамбера:

```
алг CRNUM (цел AG, AV, B, C)
рез AG, AV, B, C
нач цел X, Y, W, Z, ZN
  для X от 2 до 31
    для Y от 2 до 9
      AG := STEP (X, Y)
      если TRI (AG)
        то
            для W от 5 до 199
              если W >= 11
                то ZN := W - 9
                иначе ZN := 2
              все
              для Z от ZN до W - 3
                B := W * (Y * Z + 1)
                если TRI (B)
                  то AV := STEP (W - Z, Z) - Z * W
                     если TRI (AV)
                       то C := STEP (Y, W - Z)
                          если TRI (C)
```

```
                          то если CROSS (AG, AV, B, C)
                               то печать результата
                             все
                         все
                     все
                  все
                конец цикла по Z
              конец цикла по W
         все
       конец цикла по Y
     конец цикла по X
   конец алгоритма CRNUM.
```

**Задания**

Ж. Записать программу CRNUM на языке программирования вашей ЭВМ. Выполнив ее, узнать количество разных решений. Подсчитать (с помощью счетчика в программе), сколько четверок чисел проверяются в попытках заполнить кроссчамбер. Определить число шагов выполнения самого внутреннего цикла.

З. Составить свой кроссчамбер и предложить товарищам решить его с помощью ЭВМ.

Ниже приводится текст программы CRNUM на языке Паскаль.

```
PROGRAM CRNUM (INPUT, OUTPUT) ;
VAR X, Y, Z, W, ZN, AG, AV, B, C : INTEGER ;
(* функция проверки числа на трехзначность *)
FUNCTION TRI (A : INTEGER) : BOOLEAN ;
  BEGIN
    IF (A < = 999) AND (A > = 100)
    THEN TRI : = TRUE
    ELSE TRI : = FALSE
  END ;
(* функция проверки заполнения кросснамбера *)
FUNCTION CROSS (AG, AV, B, C : INTEGER) : BOOLEAN ;
  VAR AG1, AV1, AG3, AV3, B1, B3, C1, C3 : INTEGER ;
  BEGIN CROSS : = FALSE ;
    AG1 : = AG DIV 100 ;
    AV1 : = AV DIV 100 ;
    IF AG1 = AV1
    THEN BEGIN AG3 : = AG MOD 10 ;
               B1 : = B DIV 100 ;
               IF AG3 = B1
               THEN BEGIN B3 : = B MOD 10 ;
                          C3 : = C MOD 10 ;
                          IF B3 = C3
                          THEN BEGIN C1 : = C DIV 100 ;
                                     AV3 : = AV MOD 10 ;
                                     IF C1 = AV3
                                     THEN CROSS : = TRUE
```

```
                                    END
                          END
              END
    END ;

(* функция вычисления A в степени N *)
FUNCTION  STEP (A, N : INTEGER) : INTEGER ;
   VAR  REZ, POK, R ; INTEGER ;
   BEGIN  REZ : = 1 ;   R : = A ;   POK : = N ;
      WHILE  POK > 0   DO
         BEGIN  IF  (POK  MOD  2) < >0
                THEN  BEGIN  POK : = POK - 1 ;
                             REZ : = REZ * R
                      END;
                IF  POK >  0
                THEN  BEGIN  POK : = POK   DIV  2 ;
                             R : = R * R
                      END
         END ;
      STEP : = REZ
   END ;
BEGIN
FOR  X : = 2   TO  31  DO
   FOR   Y : = 2   TO  9  DO
      BEGIN  (* вычисление A по горизонтали *)
         AG : = STEP (X, Y) ;
         IF  TRI (AG)
         THEN
            FOR   W : = 5   TO  199  DO
               BEGIN
                  IF   W >= 11   THEN  ZN : = W - 9   ELSE  ZN : =2 ;
                  FOR   Z : = ZN   TO   W - 3   DO
                     BEGIN   (* вычисление B по вертикали* )
                        B : = W * (Y * Z + 1) ;
                     IF   TRI (B)
                     THEN
                        BEGIN   (* вычисление A по вертикали* )
                           AV : = STEP (W-X, Z) - Z * W ;
                           IF   TRI  (AV)
                           THEN
                              BEGIN   (* вычисление C по горизонтали* )
                                 C : = STEP (Y, W - Z) ;
                                 IF   TRI (C)
                                 THEN
                                    IF   CROSS (AG, AV, B, C)
                                    THEN
                                       BEGIN  (* печать   решения кросснамбера* )
                                       WRITELN ('AG=', AG, ' ', AV, 'B=', B,
                                                ' ', 'C=', C)
                                        WRITELN ('X=', X, ' ', 'Y', Y, ' ',
```

```
                                    'Z=',Z,' ','W=',W)
                           END
                        END
                     END
                  END
               END
            END
END.
```

## ТАРАБАРСКАЯ ГРАМОТА

*Отыщи всему начало и ты многое поймешь.*

***Козьма Прутков***

Цитируем [2]: "В книге М. Зуева-Ордынца "Сказание о граде Ново-Китеже" есть интересное упоминание об одном из первых русских шифров, которыми пользовались в 15–16 вв. на Руси для составления секретных бумаг в переписке царского двора с русскими послами, находившимися за границей. В наше время этот способ тайнописи кажется простым, но в те времена мало кто сумел бы, не зная ключа, прочитать письмо, зашифрованное "тарабарской грамотой" (так называли в то время тайные письмена).

Способ шифровки был следующий. Все согласные буквы русской азбуки записывались в два ряда; одна половина букв вверху, другая половина – внизу, причем в обратном порядке (одна буква под другой).

Б В Г Д Ж З К Л М Н
Щ Ш Ч Ц Х Ф Т С Р П

При зашифровке слов согласные взаимно заменялись, а гласные, Й и буквы Ъ, Ь вписывались без изменений. Слова записывались без промежутков между ними, как вообще писался любой текст до 16 в., и это еще больше затрудняло разгадывание содержания писем".

**Задача:** Написать алгоритм шифровки и дешифровки текста в соответствии с описанными у М.Зуева-Ордынца правилами.

Сначала обсудим вопрос о способах представления исходной информации. Текст, подлежащий зашифровке и дешифровке, будем рассматривать как символьный массив соответствующей длины. Назовем его OLDTEXT. Очевидно, что понадобятся еще два символьных массива:

1) массив согласных букв (назовем его SOGL);

2) массив гласных букв, в который целесообразно добавить неизменяемые при шифровке буквы: Й, Ь, Ъ (назовем его GLAS).

Значения элементов соответствующих массивов приведены ниже:

| | | |
|---|---|---|
| SOGL | БВГДЖЗКЛМНЩШЧЦХФТСРП | 20 элементов |
| GLAS | АЕЁИОУЫЭЮЯЪЬЙ | 13 элементов |

Зашифрованный или дешифрованный текст можно записывать на старом месте (в массиве OLDTEXT) или выделить для него новое место – объявить новый символьный массив с аналогичными характеристиками и с именем NEWTEXT. Таким образом, описания необходимых переменных в задаче будут иметь вид:

```
лит таб OLDTEXT [1..N]
лит таб NEWTEXT [1..N]
лит таб SOGL [1..20]
лит таб GLAS [1..13].
```

## РАЗРАБОТКА АЛГОРИТМА ШИФРОВКИ

Общий вид алгоритма очевиден:

```
алг SHIFR
  ВВОД массивов букв и шифруемого текста
  шифровка текста
  ВЫВОД зашифрованного текста
  конец алгоритма SHIFR.
```

## ДЕТАЛИЗАЦИЯ АЛГОРИТМА

Существенным предложением, подлежащим детализации, является второе – шифровка текста. Каждая буква текста должна быть просмотрена, и произведена необходимая замена. Это можно сделать, используя циклическую конструкцию. Для ее организации введем счетчик с именем I.

```
цикл по I от 1 до N.
  проверка: I-я буква не согласная?
    ДА: замены не производится
    НЕТ: произвести соответствующую замену
         буквы
  конец проверки
конец цикла по I.
```

Проверка факта, что I-я буква текста не является согласной буквой, также может быть реализована с помощью циклической конструкции, но уже с другим счетчиком. Назовем его именем J.

```
J := 1
```

**цикл пока** J < = 13

**проверка:** OLDTEXT [I] = GLAS [J] ?

ДА: выйти из цикла по J

НЕТ: произвести замену буквы

следующее значение J

**конец проверки**

**конец цикла** по J.

Теперь детализируем предложение – произвести соответствующу замену буквы. Все согласные буквы находятся в одном массиве SOG и расположены следующим образом: БВГДЖЗКЛМНЩШЧЦХФТСPI При таком расположении

1-я буква (Б) заменяется на 11-ю (Щ),

2-я буква (В) заменяется на 12-ю (Ш) и т. д.,

10-я буква (Н) заменяется на 20-ю (П) и т. д.,

20-я буква (П) заменяется на 10-ю (Н).

Таким образом, алгоритм замены I-й буквы может быть записан следующим образом:

определить номер (NOMB) буквы в массиве SOGL

**проверка:** NOMB меньше или равен 10 ?

ДА: заменить I-ю букву текста на букву SOGL [NOMB + 10]

НЕТ: заменить I-ю букву текста на букву SOGL [NOMB – 10]

**конец проверки.**

Детализируем первое предложение последнего фрагмента. Для определения номера NOMB необходимо I-ю букву текста сравнить со всеми буквами в массиве SOGL. Разумно предположить, что в записи текста может встретиться символ, который не входит ни в массив SOGL, ни в массив GLAS. В этом случае алгоритм должен отреагировать, например, сообщением "НЕИЗВЕСТНЫЙ СИМВОЛ", и либо прекратить дальнейший просмотр текста, либо оставить символ без изменений. Для определенности остановимся на последнем варианте. Следовательно, рассматриваемое предложение детализируется в виде циклической конструкции, которую оформим со счетчиком J, имеющимся от 1 до 20.

J : = 1

**цикл пока** J <= 20

**проверка:** OLDTEXT [I] = SOGL [J]

ДА: NOMB присвоить значение J

выйти из цикла по J

НЕТ: следующее значение J

**конец проверки**

**конец цикла** по J.

**проверка:** найден неизвестный символ ?

ДА: оставить символ без изменения

НЕТ: произвести замену согласной буквы

**конец проверки.**

Обратим внимание на то, что в приведенном алгоритме выход из цикла может происходить при двух разных ситуациях. Если обнаружено совпадение букв, то J будет обязательно находиться в диапазоне от 1 до 20, а иначе J будет больше граничного значения 20. Анализируя это значение после цикла, можно зафиксировать ситуацию, когда в тексте обнаружен неизвестный символ.

Теперь можно записать весь алгоритм целиком.

```
алг  SHIFR
  ВВОД шифруемого текста и массивов  SOGL и GLAS
  для  I  от  1  до  N
    J : = 1
    пока  J < = 13
      если  OLDTEXT [I] = GLAS [J]
        то J: = 15 (*для выхода из цикла*)
           NEWTEXT [I] : = OLDTEXT [I]
        иначе  J : = J + 1
       все
     конец цикла пока
     J : = 1
     пока  J < = 20
       если  OLDTEXT [I] = SOGL [J]
         то  NOMB : = J
             J : = 25  (* для выхода из цикла *)
         иначе  J : = J + 1
       все
     конец цикла пока
     если  J = 25
       то  если  NOMB > 10
                то  NEWTEXT [ I ] : = SOGL [NOMB - 10]
      иначе NEWTEXT [I] : = SOGL [NOMB + 10]
              все
       иначе  NEWTEXT [I] : = OLDTEXT [I]
     все
   конец цикла по I
   ВЫВОД зашифрованного текста
 конец алгоритма  SHIFR.
```

*УПРАЖНЕНИЯ*

1. Написать алгоритм дешифровки текста, зашифрованного описанным способом.
2. Написать алгоритм шифровки текста при условии, что сохраняется прежнее правило замены букв, но согласные буквы массива SOGL расположены следующим образом:

   Б В Г Д Ж З К Л М Н П Р С Т Ф Х Ц Ч Ш Щ.

3. Написать алгоритмы шифровки и дешифровки текста, помещая новый

текст на старое место, т.е. пользуясь в задаче только одним массивом OLDTEXT.

4. Написать алгоритмы шифровки и дешифровки текста в случае, если в добавление к согласным буквам гласные тоже заменяются по аналогичному для согласных правилу:

А Е Е И О
Я Ю Э Ы У.

5. Написать алгоритмы шифровки и дешифровки текстов для случая, когда первые 10 согласных букв заменяются на соответствующие гласные, остальные – в соответствии с таблицей:

Б В Г Д Ж З К Л М Н П Р С Т Ф
А Е Е И О У Ы Э Ю Я Щ Ш Ч Ц Х.

6. Написать алгоритм шифровки, пользуясь методом циклического сдвига букв алфавита (рис. 5).

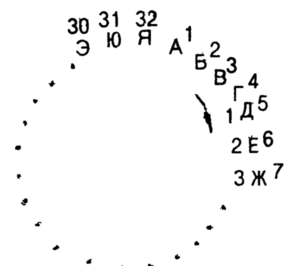



Рис. 5

7. В Некотором Интеллектуальном Институте научные сотрудники занимались конструированием программы "Робот-шифровальщик". Дни и ночи напряженного труда увенчались успехом – программа заработала. Весь институт бурлил от радости, каждый хотел получить шифровку своего текста. Но однажды чья-то коварная рука похитила дискету с программой, а черновики разработок уже были выброшены. Сохранились только образцы шифровки нескольких текстов. Один из них – стихотворение – робот превратил в строку чисел, разделенных символом "!":

Сквозь сеть алмазную зазеленел восток.
Вдаль по земле таинственной и строгой
Лучатся тысячи тропинок и дорог.
О, если б нам пройти чрез мир одной дорогой !
Все видеть, все понять, все знать, все пережить,
Все формы, все цвета вобрать в себя глазами,
Пройти по всей земле горящими ступнями,
Все воспринять и снова воплотить.

[М. Волошин]

1 6 4 8 9 6 0 5 ! 2 5 2 5 12 1 7 7 6 8 1 5 0 11 !
3 1 4 1 3 6 4 3 5 7 1 9 ! 4 3 6 3 6 3 5 3 8 3 5 3
5 7 1 4 7 6 2 4 5 8 8 3 10 1 5 9 7 27 !

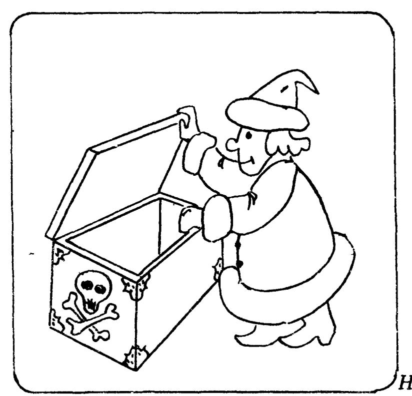
Помогите ученым восстановить алгоритм "Робота-шифровальщика".

Ниже приведена программа, написанная на языке Паскаль, для случая

шифровки по алгоритму М. Зуева-Ордынца.

```
PROGRAM  SHIFR (I, O);
TYPE  MAS = ARRAY [1..30] OF  CHAR;
      MASS = ARRAY [1..20] OF  CHAR;
      MASG= ARRAY [1..13] OF  CHAR;
VAR  NEWTEXT,  OLDTEXT : MAS;
     SOGL : MASS;  GLAS : MASG;  L, I, J, NOMB : INTEGER;
BEGIN  (* заполнение массивов GLAS и SOGL*)
  SOGL [1] := ' Б '; SOGL [2] := ' В '; SOGL [3] := ' Г '; SOGL [4] := 'Д ';
  SOGL [5] := ' Ж '; SOGL [6] := ' З '; SOGL [7] := ' К '; SOGL [8] := 'Л ';
  SOGL [9] := ' М'; SOGL[10]:= ' Н '; SOGL [11] := ' Щ'; SOGL [12]:= 'Ш ';
  SOGL [13]:= ' Ч '; SOGL [14]:= ' Ц '; SOGL [15]:= ' Х '; SOGL [16]:= ' Ф ';
  SOGL [17]:= ' Т '; SOGL [18]:= ' С '; SOGL [19]:= ' Р '; SOGL[20]:= 'П ';
  GLAS [1] := ' А '; GLAS [2] := ' Е '; GLAS [3] := ' Ё '; GLAS [4] := ' И ';
  GLAS [5] := ' О'; GLAS [6] := ' У '; GLAS [7] := ' Э '; GLAS [8] := ' Ю ';
  GLAS [9] := ' Я '; GLAS [10]:= 'Ы '; GLAS [11] := ' Ь '; GLAS [12]:= ' Ъ ';
  GLAS [13] := 'Й ';
  WRITELN ('ВВЕДИТЕ ТЕКСТ ИЗ ТРИДЦАТИ СИМВОЛОВ');
READLN
FOR I:=1 TO 30 DO BEGIN READ (OLDTEXT [I]); WRITE (OLDTEXT [I]); END;
FOR I:=1 TO 30 DO BEGIN
  BEGIN
    WRITELN;  WRITELN ('I =', I);
    J:=1;
     WHILE J <= 13  DO BEGIN
     WRITELN (' J = ', J ,'OLDTEXT [I]=', OLDTEXT [I], 'GLAS [J]= ', GLAS [J]);
        IF OLDTEXT [I] = GLAS [J]
        THEN J:= 15
        ELSE  J:= J + 1;  END;  WRITELN  ('ПОСЛЕ ЦИКЛА J =', J);
      IF  J = 15
      THEN BEGIN NEWTEXT [I]:= OLDTEXT [I]; WRITELN (NEWTEXT [I]) END
      ELSE
         BEGIN  L:= 1;
               WHILE  L <= 20  DO
               IF OLDTEXT [I] = SOGL [L]
               THEN BEGIN  NOMB := L;  L:= 25  END
               ELSE  L:= L + 1;
            IF  L <> 25
            THEN  BEGIN
                     WRITELN  ('НЕИЗВЕСТНЫЙ СИМВОЛ');
                     NEWTEXT [I] := OLDTEXT [I]
                  END
             ELSE  IF NOMB > 10
                  THEN  NEWTEXT [I] := SOGL [NOMB - 10]
                  ELSE  NEWTEXT [I] := SOGL [NOMB + 10];
          END
  END; END; WRITELN; FOR I:= 1 TO 30 DO WRITE (NEWTEXT [I]);
END.
```

# КОМПЬЮТЕР ДЛЯ ВИЛЬЯМА ЛЕГРАНА



*Немало сокровищниц в мире открывается, как Сезам сказочного Али-Бабы, словесным ключом.*

***Ван-Дейк***

"...Легран разогрел пергамент и дал его мне. Между черепом и козленком, грубо начертанным чем-то красным, стояли такие знаки: 53 # # + 305 )) 6★ ...

– Что ж! – сказал я, возвращая Леграну пергамент, – меня это не подвинуло бы ни на шаг. За все алмазы Голконды я не возьмусь решать подобную головоломку.

– И все же, – сказал Легран, – она не столь трудна, как может сперва показаться. Эти знаки, конечно, – шифр; иными словами, они скрывают словесную запись. Кидд, насколько мы можем о нем судить, не сумел бы составить истинно сложную криптограмму. И я сразу решил, что передо мной примитивный шифр, но притом такой, который незатейливой фантазии моряка должен был показаться совершенно непостижимым.

– И что же, вы сумели найти решение ?

– С легкостью! В моей практике встречались шифры в тысячу раз сложнее..."

Что же сделал Легран, чтобы разгадать криптограмму ?

Проследим за его действиями. Сначала он путем логических рассуждений определил, что текст мог быть написан только на английском языке. Этот текст шел сплошной строкой, без просветов, и потому Легран начал подсчитывать однотипные знаки, чтобы узнать, какие из них чаще, а какие реже встречаются в криптограмме. В результате получилась таблица, в которой на первом месте по частоте появления оказался знак 8. Поскольку Легран знал, что в английской письменной речи самая частая буква E, он предположил, что этот знак и есть шифр буквы E. Он убедился в этом, исходя из еще одной характеристики английской письменной речи: буква E в словах часто удваивается. Оказалось, что в короткой криптограмме удвоение знака 8 встречается пять раз.

Далее Легран воспользовался тем, что самое частое слово в английском языке – артикль THE. И он проанализировал криптограмму, отыскивая в ней сочетания из трех знаков, оканчивающихся знаком 8. Таким образом, он предположительно определил буквы Т и Н.

Мы отсылаем читателя к рассказу Эдгара По "Золотой жук" и приступаем к формулировке возможных **задач** анализа некоторого текста и к составлению соответствующих алгоритмов.

1. В заданном тексте заменить один символ другим.

2. Для заданного текста получить таблицу, состоящую из всех разных символов, встречающихся в этом тексте. Подсчитать число этих разных символов.

3. Написать алгоритм, определяющий, сколько раз встречается в тексте каждый отдельный символ.

4. Упорядочить по убыванию информацию, полученную в задачах 2 и 3.

5. В заданном тексте определить количество пар совпадающих символов. В этой задаче можно не различать пары символов, а можно составить таблицу, аналогичную таблице задачи 2. Например, пара 88 встречается 5 раз, а пара ;; – 2 раза.

6. Выделить в заданном тексте сочетание из трех заданных символов и модифицировать текст следующим образом: после каждого такого сочетания вставить в текст символ пробела.

7. Определить, какой символ встречается в тексте чаще всего (реже всего).

8. Определить, сколько в заданном тексте слов (предложений).

Обсудим теперь вопрос о типах данных, которые должен " уметь" обрабатывать компьютер для Леграна.

Текст криптограммы можно представить в виде массива символов:

<u>лит</u> <u>таб</u> ТЕКСТ [1 .. N].

Во второй и последующих задачах потребуются еще два массива. Один – для хранения <u>различных</u> символов, встречающихся в тексте криптограммы. Назовем его SIM. Необходимо зарезервировать M элементов в этом массиве, где M – максимальное число различных символов, которыми может воспользоваться человек для записи каких-либо текстов. Это достаточно большое число, если учесть, например, количество китайских иероглифов, которые могут быть применены для шифровки. Мы ограничимся в нашей задаче числом различных символов, имеющих определенные коды в компьютере. Это число, как правило, не более 128. Так, массив SIM можно описать следующим образом:

<u>лит</u> <u>таб</u> SIM [1 .. 128].

Для хранения частот появления этих символов в тексте потребуется арифметический массив той же размерности:

<u>вещ</u> <u>таб</u> KOL [1 .. 128].

Выбор характеристик остальных данных будем обсуждать по мере их появления и по мере надобности.

РАЗРАБОТКУ алгоритмов начнем с самого простейшего из сформулированных выше.

Алгоритм "Замена символа в тексте"

В. Легран, определив наиболее часто встречающийся символ в тексте (это был символ 8), заменил его на букву Е. Разгадав шифры других букв, он также производил соответствующие замены. Таким образом, эта задача весьма важна для построения общего алгоритма дешифровки. По-видимому, она не вызывает каких-либо трудностей. Необходимо посимвольно просмотреть весь текст, проверяя для каждого символа, равен он или нет заданному значению. Обозначим старое значение символа именем OLD, а новое, которое необходимо представить вместо старого, – NEW. Значения обоих имен известны. Например, символ 8 надо заменить символом Е. Для определенности предположим, что в тексте – 100 символов.

```
алг ZAMENA
  лит таб TEXT[1..100]
  лит OLD, NEW
  цел I
начало алгоритма
  ВВОД значений OLD, NEW и массива TEXT
  цикл по I от 1 до 100
    проверка: символ TEXT[I] = OLD
      ДА: заменить символ TEXT[I] на значение NEW
    конец проверки
  конец цикла по I
конец алгоритма ZAMENA.
```

**Задача 2.** Определить все различные символы, входящие в текст криптограммы (Легран составил таблицу из таких символов и частот соответственно).

Как уже отмечалось выше, в этой задаче нам потребуется массив SIM, в котором и будут помещаться все различные символы текста.

РАЗРАБОТКА АЛГОРИТМА

Очевидно, что необходим просмотр всех символов текста, т.е. основным действием в алгоритме будет цикл по всем элементам массива TEXT. Организуем этот просмотр с помощью счетчика I.

```
алг RAZSIM
  лит таб TEXT[1..100]
  лит таб SIM [1..128]
  цел I, GR (* GR – для подсчета символов в SIM *)
начало алгоритма
  ВВОД текста TEXT
  цикл по I от 1 до 100
    обработка I-го символа текста
```

**конец цикла** по I
конец алгоритма RAZSIM.

ДЕТАЛИЗИРУЕМ тело цикла.

В чем заключается обработка I-го символа? Для ответа на этот вопрос обратимся к исходным данным. Пусть, например, они имеют вид, представленный ниже

TEXT [1 .. 18] ОТЫЩИ ВСЕМУ НАЧАЛО

Тогда в массив SIM должны быть занесены следующие символы:

ОТЫЩИ ВСУМУНАЧЛ

Очевидно, что первая буква текста, безусловно, должны быть в массиве SIM. Занесение первой буквы в этот массив должна быть выполнено до цикла по I (**инициализация цикла по** I), поскольку в цикле предполагается сравнить буквы текста с уже попавшими в массив SIM символами. Это требуется условием задачи: в массиве дожны быть только разные символы.

Таким образом, до цикла по I в элемент SIM [1] помещается первый символ текста и переменная GR, указывающая, сколько элементов массива SIM занято, принимает значение 1. Просмотр элементов массива TEXT теперь следует вести со второго элемента.

GR : = 1 ; SIM [GR] : = TEXT [1]
**цикл** по I от **2** до 100
**проверка:** I-й символ текста совпадает с одним из символов массива SIM
ДА: прекращение просмотра массива SIM
НЕТ: занесение символа TEXT [I] в массив SIM увеличение значения GR на 1
**конец проверки**
**конец цикла** по I.

Приведенная запись достаточно понятна и не нуждается в дополнительных комментариях, поэтому можно приступить к обсуждению вопроса, как реализовать **проверку** совпадения символов в массивах TEXT и SIM. Ясно, что символ TEXT [I] надо сравнить со всеми символами в массиве SIM, т.е. мы имеем дело с циклической конструкцией. Организуем ее с помощью счетчика с именем J, значение которого должно изменяться от 1 до значения переменной GR. Возможны два случая. Остановимся на них подробнее.

1. Рассматриваемый элемент текста **не совпадает** ни с одним из символов в массиве SIM. В этом случае массив SIM будет просмотрен до конца и конечное значение счетчика J будет равно значению выражения GR + 1. Здесь важно подчеркнуть, что мы можем гарантировать это значение счетчика только в том случае, если сами управляем его изменением. Иными словами, в подобной ситуации необходимо пользоваться только циклом **ПОКА**.

2. При просмотре массива SIM обнаруживается **совпадение** символов. Например, 12-й элемент текста (пример см. выше), являющийся символом

пробела, совпадает с 6-м элементом массива SIM, который в этом случае не должен просматриваться до конца, т.е. необходимо обеспечить выход из цикла по J в момент обнаружения совпадения символов. При использовании цикла **ПОКА** это можно сделать присвоением J значения, заведомо большего граничного. Например, в нашем случае J можно присвоить значение GR + 10, что больше значения выражения GR + 1. Таким образом, имеем

```
J : = 1
цикл пока J меньше или равно значению GR
  проверка: TEXT [I] = SIM [J]
    ДА:   J : = GR + 10
    НЕТ:  J : = J + 1
  конец проверки
конец цикла пока.
```

Проверяя далее в алгоритме значение счетчика J, можно судить о том, имело место совпадение символов или нет:

значение J равно значению выражения GR + 10 – имело;

значение J не равно значению выражения GR + 10 – не имело.

Описанную ситуацию обработки I-го символа в цикле по J можно представить в виде другого алгоритма.

Введем некоторую дополнительную переменную арифметического (например целого) типа с именем, например L. Пусть она способна принимать только два значения – 0 и 1. Это, безусловно, не обязательно; эти значения могут быть, например, 3 и 1000, но их может быть только два. До входа в цикл присвоим L значение 0 и договоримся изменить его на 1 только в случае, если возникает совпадение символов массива TEXT и SIM. Если проверять значение переменной L в заголовке цикла, то с помощью этого значения можно управлять процессом выполнения тела цикла:

L = 0 – тело цикла продолжает выполняться (нет совпадения),

L = 1 – происходит выход из цикла (есть совпадение).

Отметим, что проверка на граничное значение числа элементов в массиве SIM в этом случае остается прежней, так как необходимо просмотреть конечное число элементов (GR) массива SIM. Таким образом, можно записать следующий вариант:

```
. . .
цел L
. . .
цикл пока (L = 0) И (J < = GR)
  проверка: TEXT [I] = SIM [J]
    ДА: L : = 1
    НЕТ : J : = J + 1
  конец проверки
конец цикла пока.
```

В заголовке цикла в данном случае написано сложное ЛОГИЧЕСКОЕ выражение, проверяющее совместное выполнение двух событий –

равенство значения переменной L нулю **И** непревышение счетчиком J граничного значения GR. Когда оба эти события выполняются одновременно, логическое выражение **истинно** (справедливо). При невыполнении хотя бы одного из них логическое выражение становится **ложным** (не справедливо) и обеспечивает выход из цикла. При программировании подобных ситуаций переменную L обычно описывают не как целую, а как ЛОГИЧЕСКУЮ переменную и те два значения, которые она может принимать, трактуют как ИСТИНА (английское слово – TRUE) и ЛОЖЬ (английское слово – FALSE). В задаче "Расписание уроков" логические переменные описаны подробно. Нам же достаточно сказанного для того, чтобы записать вариант алгоритма.

```
. . .
лог L
. . .
L := ИСТИНА,   J := 1
цикл пока ( L ) И (J <= GR)
   проверка: TEXT [I] = SIM [J]
      ДА:  L := ЛОЖЬ
      НЕТ: J := J + 1
   конец проверки
конец цикла пока.
```

В нашем примере не видно особых преимуществ описанного способа записи алгоритма, но для задач, где имеется много альтернатив для выхода из цикла, и не требуется их раздельная обработка после выхода, – такой способ может оказаться более компактным.

**Задания**

А. Придумайте пример задачи, где наиболее приемлем способ организации цикла с логической переменной.

Б. Можно ли в нашей задаче обойтись без сложного логического условия в заголовке цикла, а именно – изменять значение на ЛОЖЬ и в том случае, если достигается граничное значение GR ?

**Задача 3** – написать алгоритм, подсчитывающий, сколько раз встречается в тексте каждый отдельный символ.

Составим требующийся алгоритм, используя предыдущий.

В случае **задачи 2** при просмотре исходного текста формировался массив различных символов SIM. Всякий раз, когда встречался символ, уже находящийся в массиве SIM, в этом алгоритме совершался переход к анализу следующего символа текста, и факты, какие символы и который раз совпали, нигде и никак не фиксировались. Это как раз тот момент, который отличает **задачу 3** от **задачи 2**. В рассматриваемой задаче в подобном случае надо увеличить на 1 число вхождений данного символа в текст.

Для хранения числа вхождений различных символов в исходный текст (частоты их появления в нем) в **задаче 3** необходимо объявить целый массив (KOL) с той же размерностью, что и SIM.

Ниже приведено окончательное расположение информации в массивах TEXT, SIM и KOL для конкретного примера.

ЫЩИ ВСЕМУ НАЧАЛО И ТЫ МНОГОЕ ПОЙМЕШЬ. TEXT [1..39]
ЫЩИ ВСЕМУНАЧЛГПЙЕШЬ. SIM [1..22]
2126112312211111111 KOL [1..22]

**Задание Г.** Напишите алгоритм для **задачи 3,** вчеся следующие олнения в алгоритм для **задачи 2:**

1. Описание массива KOL.

2. Занесение начальной информации в массивы KOL (обнуление ментов) и SIM. В последнем случае надо подумать над тем, какой вид жна иметь начальная информация в массиве SIM. Этот момент важен ля задачи 2.

3. Увеличение на 1 значения соответствующего элемента массива L при повторных встречах соответствующего символа.

**Задача 4** – поиск максимальной частоты появления символа; упоряіение таблицы.

Вильям Легран после анализа текста криптограммы получил таблицу, которой все символы расположены в порядке убывания частоты их явления в тексте (см. рассказ "Золотой жук" Э. По). Подобная задача сит название задачи <u>сортировки</u> или <u>упорядочения</u>. Алгоритмов соровки очень много. Н. Вирт пишет, что весь курс программирования жно было бы построить на базе изучения алгоритмов сортировки. тодическая ценность рассматриваемой задачи в том, что она является только базой для изучения алгоритмов работы с символьной информцией, но и обеспечивает интерес к сложным алгоритмам упорядочения блиц.

В приложении имеется лист опорных сигналов по простейшим оритмам сортировки. В этом разделе мы приводим очень интересный оритм, разработанный Н. Виртом и относящийся к типу сортировки авками, названный им "сортировкой с просеиванием". Мы предлагаем агмент из книги Н. Вирта [7] и рекомендуем в качестве приятного ажнения разобраться в алгоритме, написанном программистом сшего класса.

"<u>Сортировка простыми включениями</u>

Этот метод используют игроки в карты. Элементы (карты) условно зделяются на готовую (упорядоченную) последовательность ,..., $A_{i-1}$ и входную последовательность $A_i, \ldots, A_n$. На каждом шаге, иная с $i = 2$ и увеличивая $i$ на 1, берут $i$-й элемент входной последоельности и передают в готовую последовательность, вставляя его на дходящее место...

Алгоритм сортировки простыми включениями выглядит следующим разом:

```
FOR I:=2 TO N DO
  BEGIN
    X:=A[I];
    вставить X на подходящее место в A1,...,Ai
  END.
```

При поиске подходящего места удобно чередовать сравнения и пересылки, т.е. как бы "просеивать" X, сравнивая его с очередным элементом $A_j$ и либо вставляя X, либо пересылая $A_j$ направо и продвигаясь налево. Заметим, что "просеивание" может закончиться при двух различных условиях:

1. Найден элемент $A_j$, меньший, чем X.
2. Достигнут левый конец готовой последовательности.

Этот типичный пример с двумя условиями окончания дает нам возможность рассмотреть хорошо известный прием фиктивного элемента ("барьера"). Его можно легко применить в этом случае, установив барьер $A_0$ = X. Заметим, что для этого нужно расширить диапазон индексов в описании A до [0 . . $n$]." Окончательный алгоритм представлен в виде программы *:

```
PROCEDURE SITO ;
VAR X, I, J
BEGIN
   FOR I := 2 TO N DO
      BEGIN
         X := A [ I ] ;  A [0] := X ;  J := I - 1 ;
         WHILE X <A [J] DO
            BEGIN
               A [J + 1] := A [J] ;  J := J - 1
            END ;
         A [J + 1] := X
      END
END.
```

*УПРАЖНЕНИЯ*

Предлагаем в качестве упражнений выполнить все сформулированные выше задания, а также множество других, которые покажутся вам целесообразными для того, чтобы оказать помощь Вильяму Леграну.

Ниже приведено несколько программ на языке Паскаль для разработанных алгоритмов.

```
PROGRAM ZAMENA (INPUT, OUTPUT) ;
(* программа, производящая замену символа в тексте *)
CONST  N = 38 ;  M = 128 ;
TYPE MAS1 = ARRAY [1 .. N]  OF  CHAR ;
VAR TEKCT : MAS1 ;
      I : INTEGER ;
      OLD, NEW : CHAR ;
BEGIN
  WRITELN ('ВВЕДИТЕ ЗНАЧЕНИЯ СИМВОЛОВ ДЛЯ ЗАМЕНЫ–СТАРОГО И НОВОГО');
  READLN (OLD, NEW) ;
```

* В приведенной программе авторы изменили имена переменных и несколько упростили описания, чтобы сделать ее текст более близким к используемым ими обозначениям. Массив A – глобальный.

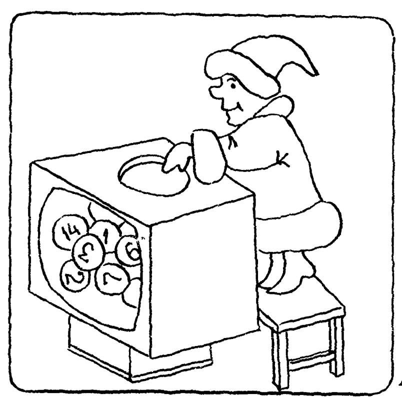
```
  WRITELN (' ВВЕДИТЕ ТЕКСТ');
  FOR I := 1 TO N DO READ (TEKCT [I]);
  FOR I := 1 TO N DO
    IF TEKCT [I] = OLD
    THEN TEKCT [I] := NEW
    ELSE ;
  WRITELN (' модифицированный текст');
  FOR I := 1 TO N DO WRITE (TEKCT [I] )
END.
```

```
PROGRAM RAZSIM (INPUT, OUTPUT);
(* программа, подсчитывающая число различных символов в тексте *)
CONST N = 38; M = 128;
TYPE MAS = ARRAY [1..N] OF CHAR;
     MASS = ARRAY [1..M] OF CHAR;
VAR TEXT : MAS;
    SIM : MASS;
    L : BOOLEAN;
    I, J, K, GR : INTEGER;
BEGIN
  WRITELN (' ВВЕДИТЕ ТЕКСТ');
  READLN;
  FOR I := 1 TO N DO READ (TEXT [I]);
  FOR I := 1 TO M DO SIM [I] := ' ';
  GR := 1; SIM [GR] := TEXT [1];
  FOR I := 1 TO N DO
    BEGIN
      J := 1; L := TRUE;
      WHILE (J <= GR) AND L DO
        IF TEXT [I] = SIM [J]
        THEN L := FALSE
        ELSE J := J + 1;
      IF L (* L – ИСТИНА, т.е. этот случай соответствует ситуации
              появления нового символа *)
      THEN
        BEGIN GR := GR + 1; SIM [GR] := TEXT [I] END
    END;
  WRITELN (' число различных символов в тексте – ', GR : 6);
  WRITELN (' различные символы, встречающиеся в тексте:');
  FOR I := 1 TO GR DO WRITE (SIM [I] )
END.
```

```
PROGRAM HOWMANY (INPUT, OUTPUT),
(* программа, подсчитывающая число различных символов в тексте *)
(* и количество его вхождений в текст *)
CONST N = 38; M = 20;
TYPE MAS = ARRAY [1..N] OF CHAR;
     MASS = ARRAY [1..M] OF CHAR;
```

```
VAR TEXT : MAS ;
      SIM : MASS ;
      KOL : ARRAY [1 . . M]  OF  INTEGER ;
      L : BOOLEAN ;
      I, J, K, GR : IMTEGER ;
BEGIN
  WRITELN  ('ВВЕДИТЕ ТЕКСТ');
  READLN ;
  FOR  I : = 1  TO  N  DO  READ (TEXT [I] ) ;
  FOR  I : = 1  TO  M  DO  BEGIN SIM [I] : = '  '; KOL [I] : = 0  END ;
  GR . = 1 ;  SIM [GR] : = TEXT [1];  KOL [GR] : = 1 ;
FOR  I : = 2  TO  N  DO
  BEGIN
    J : = 1 ;  L : = TRUE ;
    WHILE  (J  < = GR)  AND  L  DO
       IF  TEXT [I] = SIM [J]
       THEN
          BEGIN
             L : = FALSE ;
             KOL [J] : = KOL [J] + 1
          END
       ELSE  J : = J + 1 ;
    IF  L  (* L – ИСТИНА, т.е. этот случай соответствует ситуации появления
             нового  символа *)
    THEN
       BEGIN  GR:=GR+1;  SIM [GR] : = TEXT [I] ;  KOL [GR] : = 1  END
  END ;
  WRITELN ('число различных символов в тексте –', GR : 6 ) ;
  WRITELN ('различные символы, встречающиеся в тексте  ') ;
  FOR  I : = 1  TO GR  DO  WRITE (SIM [I] ) ;  WRITELN ;
  WRITELN ('    СИМВОЛ       КОЛИЧЕСТВО') ;
  FOR  I  : = 1  TO  GR  DO  WRITELN (SIM [I] : 6,  KOL [I] : 14 ) ;
END
```

# СПОРТЛОТО



*Англичанин никогда не шутит, если речь идет о такой важной вещи, как пари.*

***Ж. Верн***

Принимали ли вы когда-нибудь участие в розыгрыше подписки на собрание сочинений вашего любимого автора ? Или, может быть, вы заядлый любитель денежно-вещевой лотереи, мечтающий выиграть видеомагнитофон или персональный компьютер? (Кстати, а почему бы действительно не разыгрывать в лотереях персональные ЭВМ ?). А "Спортлото", "Спринт", книжная лотерея ? Сколько соблазнов и искушений окружает нас ! Наверное, в каждом человеке живет искорка азарта, надежда на везение, желание попытать счастья. Потому так популярны игровые автоматы, а компьютерные классы всегда переполнены "хроническими" игроками. Но мы считаем, что гораздо занимательнее и интереснее самим составлять игровые программы, учить компьютер новым играм. При этом "убиваются два зайца": во-первых, программист совершенствует свое профессиональное мастерство, испытывает высокое интеллектуальное удовлетворение в установлении контакта с "думающей" машиной, производит новый программный продукт, который интересен многим, во-вторых, когда программа готова, он может и сам с нею поиграть.

Составление игровых программ – это особое искусство. При этом помимо реализации стратегии выбранной игры нужно уделять максимум внимания изобразительным средствам (графика, мультипликация, цвет) и звуковому сопровождению, т.е. для получения интересной программы придется досконально разобраться в возможностях вашего персонального компьютера.

Чтобы познакомиться с некоторыми вычислительными приемами, часто используемыми в игровых алгоритмах, мы будем ориентироваться на ЭВМ с алфавитно-цифровым дисплеем. А **ЗАДАЧЕЙ** для себя поставим научить ЭВМ разыгрывать тираж "Спортлото 5 из 36" и проверять заданные нами числа. Условия этой игры всем хорошо известны, поэтому

описывать их не будем. Единственное, чего нельзя пообещать игрокам, это материального вознаграждения от ЭВМ. Но, с другой стороны, можно вознаградить себя морально, играя много-много раз.

Итак, во-первых, нужно составить алгоритм, в котором загадываются, иначе говоря, создаются 5 разных чисел, значения которых никак не связаны друг с другом, а величина лежит в интервале от 1 до 36. Причем любое из чисел этого интервала может появиться в тираже с одинаковой вероятностью. И, естественно человек-игрок не должен знать, какие это числа, а также алгоритм их получения, иначе по первому числу он сможет определить остальные. В таких случаях на помощь приходит специальная функция, которая вырабатывает так называемые случайные числа, в Турбопаскале это функция RANDOM (X). О датчиках случайных чисел мы говорили подробно в задаче "Арифметика для малышей".

Учитывая эту информацию, можно приступить к составлению алгоритма генерации тиража. Но, как всегда, подумаем сначала о типе данных, с которыми мы собираемся работать. Первая мысль – создать 5-элементный массив и заносить туда получаемые числа. Это обыкновенный путь решения. Правда, если ваша ЭВМ работает с языками программирования, в которых других подходящих типов нет, то именно на массиве придется остановиться. В [43] приводится вариант решения такой задачи. Мы же хотим на примере этой задачи познакомить читателя с новым типом данных – множеством.

Определение. Множество – это неупорядоченная совокупность не совпадающих между собой элементов, которые имеют некоторое общее свойство, позволяющее рассматривать их совместно. Например, множество натуральных чисел, множество женских имен, множество целых чисел от 1 до 36.

Изучением объектов такой природы занимается теория множеств, в которой нет никаких ограничений на типы объектов, составляющих множество. Мы познакомимся с алгоритмическим воплощением подобных данных на примере языка программирования Турбопаскаль, в котором имеются определенные ограничения на использование данных множественного типа. Объектами множества могут быть данные скалярного (кроме целого и вещественного), интервального (от ... до) или перечисляемого типов. Число элементов во множестве не должно превышать 256.

Для описания множественного типа в Турбопаскале используется служебное слово SEN. Так же, как при описании массивов, нужно указать тип элементов множества; это можно сделать заранее или непосредственно, описывая множество. Например, описание переменной множественного типа со значениями от 1 до 36 :

```
TYPE  WID = 1 .. 36 ;  (* тип элементов множества * )
      KART = SET OF WID ;
VAR   ELEM : WID ;
      POSL, MNO : KART.
```

Переменная ELEM может быть равна 1, или 2, или 15, или 36, одному из чисел указанного интервала. Значением же переменной POSL является любая совокупность разных чисел из указанного интервала. Элементы,

являющиеся значением множественной переменной, записываются в квадратных скобках через запятую в любом порядке. Таким образом, POSL может быть равна [1, 2, 3, 5, 7, 12, 36], или [4, 7, 1, 20], или [35, 2], или [ ] – пустому множеству, не содержащему никаких элементов.

Основным отличием данных множественного типа от массивов является невозможность перенумеровать их элементы: для элемента массива можно указать его номер в совокупности, а для элемента множества – лишь присутствие или отсутствие его в значении множественной переменной.

В языке Турбопаскаль разрешены следующие действия над данными множественного типа:

1) присваивание

POSL := [2, 4, 6]; POSL := [ ];

2) отношение Имеется 4 операции отношения:

| | |
|---|---|
| = | равенство множеств, |
| <> | неравенство множеств, |
| <=, => | включение множеств. |

Существует дополнительная операция IN, которая введена для определения членства во множестве. Например, если ELEM=5, а POSL = =[1, 2, 3, 4, 5, 6], то выражение ELEM IN POSL будет равно ИСТИНА. Для POSL =[2, 4, 6, 8] то же выражение равно ЛОЖЬ.

3) объединение. Знак операции +. Пусть POSL = [1, 3, 5], а MNO = =[2,4, 6]. После выполнения POSL : = POSL + MNO будем иметь POSL = =[1, 2, 3, 4, 5, 6]. Если ELEM = 5,а POSL =[2, 4, 6, 7], то после выполнения присваивания POSL : = POSL + [ELEM] значение POSL изменится – [2, 4, 5, 6, 7]. Следовательно, операция объединения позволяет добавить новый элемент во множество.

4) разность. Знак операции –. Пусть POSL =[1, 2, 3, 4, 5, 6], MNO = =[2, 4, 6]. Тогда после выполнения POSL : = POSL – MNO POSL будет равно [1, 3, 5]. Если ELEM = 5,а POSL =[4, 5, 6], то после POSL : = POSL – [ELEM] будет POSL = [4, 6].

5) пересечение. Знак операции ★ . Пусть POSL =[2, 4, 6, 8], а MNO = = [1, 4, 7]. Тогда после POSL : = POSL ★ MNO имеем POSL = [4].

*УПРАЖНЕНИЯ*

1. Для проверки равенства двух множественных переменных достаточно записать одно логическое выражение. А как проверить равенство элементов двух массивов, учитывая, что равные значения могут иметь элементы с разными номерами? Решить противоположную задачу – проверить массивы на неравенство.
2. Составить алгоритмы, которые реализуют для массивов операции, аналогичные действиям над множествами, – объединение, пересечение, разность.

В качестве примера рассмотрим алгоритм Эратосфена для нахождения простых чисел в заданном диапазоне от 1 до N, для которого известным ученым в области программирования К. Хоаром составлена

программа с использованием множественного типа. Идея алгоритма Эратосфена содержится в его названии – РЕШЕТО. Сначала все натуральные числа помещаются в решето (множество). Затем выбирается очередное число, начиная с 2. Это число считается простым. Затем из решета удаляются все числа, кратные 2, то есть 4, 6, 8 и т.д. Эти действия повторяются до тех пор, пока не будет просмотрено все множество.

Ниже приведена программа, отыскивающая простые числа, не превосходящие 100.

```
PROGRAM ERAT (INPUT, OUTPUT);
CONST  N = 100 ;
TYPE MNOSH = SET  OF 2 .. N ;
VAR PROST, RESHETO : MNOSH ;
    NEXT, J : INTEGER ;
BEGIN WRITE ( ' простые числа – это : ' ) ;
  RESHETO : = [2 .. N] ;
  PROST : = [  ] ;  NEXT : = 2 ;  WRITE (NEXT) ;
  REPEAT
    WHILE NOT (NEXT IN RESHETO) DO
      NEXT : = SUCC (NEXT) ;
     WRITE (NEXT);
     PROST : = PROST + [NEXT] ;  J : = NEXT ;
     WHILE J < = N DO
       BEGIN RESHETO : = RESHETO – [ J] ;
             J : = J + NEXT
       END ;
     UNTIL RESHETO = [  ]
 END.
```

В этой программе использован цикл REPEAT – UNTIL, так называемый цикл с пост-условием. Правило его работы: выполняется тело цикла, проверяется условие, записанное после UNTIL. Если оно ЛОЖЬ, то снова выполняется тело цикла. И так до тех пор, пока условие не станет ИСТИНА, после чего цикл завершается.

Возвратимся к "Спортлото 5 из 36". Любая компьютерная игра предполагает наличие диалога между ЭВМ и пользователем, хотя бы самого примитивного. Поскольку в организации такого диалога программист может пофантазировать, будем просто указывать в алгоритме тему текстов.

## РАЗРАБОТКА АЛГОРИТМА

Запишем самую общую схему алгоритма:

```
алг  SPLOT
   описания данных
начало  алгоритма
  ВЫВОД: приглашение к игре
  розыгрыш тиража "Спортлото" в памяти ЭВМ
  игрок загадывает свои номера
```

определение результатов угадывания
ВЫВОД: итоги игры
конец алгоритма SPLOT.

Как удобнее представить в памяти ЭВМ итоги розыгрыша – массивом или множеством? Нужно заглянуть немного вперед: а что мы собираемся с этими данными делать? Нам нужно будет узнать, есть ли среди этих чисел те, что загадал игрок. Если совокупность – массив, то придется каждый его элемент сравнивать с каждым загаданным числом. А если совокупность – множество, то для каждого загаданного числа достаточно применить операцию IN. Такое сокращение действий в алгоритме очень привлекательно, поэтому и остановимся на множественном типе.

Детализируя алгоритм, переносим внимание только на действие "розыгрыш тиража". Его относительная независимость от остальных частей алгоритма наводит на мысль, что здесь можно воспользоваться вспомогательным алгоритмом, а именно процедурой, так как в результате мы должны получить совокупность значений. Тираж разыгрывается случайным образом. Возьмем на вооружение датчик случайных чисел RANDOM (X) и сразу отметим, что для действительно случайного выбора числа нужно получить значение X, желательно, чтобы оно было разным при каждом новом выполнении программы. Здесь открывается простор для фантазии программиста. Мы пойдем по довольно простому пути: попросим игрока задать машине любое целое число, сделаем его входным параметром процедуры, а дальше она сама будет формировать тираж.

Процедура должна выдать множество из 5 целых чисел, взятых из интервала 1...36. Поскольку числа выпадают случайно, нужно каждое из них проверить на несовпадение с другими. И здесь нам опять поможет операция IN. Случайные числа определяются одно за другим, пока не наберется 5 разных. По-видимому, понадобится цикл типа пока.

Так как в учебном алгоритмическом языке нет множественного типа, несколько отойдем от строго формальной записи алгоритма, внеся в нее вполне понятные читаталю описания:

```
алг TIRASH ( цел X, множество из [1:36] POSL)
арг X
рез POSL
нач цел I
    цел из [1:36] KLET
    I := 0 ;  POSL := [ ]
    пока I < 5
      KLET := случайное число
      если KLET  нет в POSL
        то   POSL := POSL + [KLET]
             I := I + 1
        все
      конец цикла пока
  конец алгоритма TIRASH.
```

Как получить случайное число из заданного интервала, мы уже знаем из задачи "Арифметика для малышей". Условие "KLET нет в POSL" запишется в виде логического выражения не (KLET IN POSL). Надо сказать заранее, что такое же условие можно будет применить при определении результатов угадывания числа игроком.

Вернемся опять к общей схеме алгоритма. Нам осталось обсудить два действия: "игрок загадывает свои номера" и "определение результатов угадывания". Они останутся относительно независимыми друг от друга, если мы будем хранить в памяти ЭВМ все числа, загаданные игроком. Но такое долговременное хранение ничем не оправдано, поэтому можно объединить эти действия в последовательность "загадал число – проверка", которая выполняется 5 раз. Можно ввести все числа сразу, а ЭВМ будет считывать их по мере необходимости. Результатом проверки может быть количество угаданных чисел или сами эти числа, мы выберем первое. Тогда получим следующий алгоритм игры в "Спортлото 5 из 36":

```
алг SPLOT
    описания данных
начало алгоритма
   ВЫВОД: приглашение к игре
   TIRASH (X, POSL)
   ВЫВОД: загадать числа
   ВВОД: загаданные числа
   I : = 0
   для J от 1 до 5
      чтение числа в переменную PROBA
      если PROBA IN POSL
        то I : = I + 1
        все
     конец цикла по J
     ВЫВОД I
  конец алгоритма SPLOT.
```

*УПРАЖНЕНИЯ*

3. Что нужно изменить в программе SPLOT, чтобы она превратилась в "Спортлото 6 из 49" ?
4. Изменить программу SPLOT так, чтобы она выполнялась столько раз, сколько заранее запросит игрок.
5. Изменить программу SPLOT так, чтобы она выполнялась до тех пор, пока игрок ее не остановит.

На этом разработку алгоритма можно было бы закончить, но мы пока еще не обратили внимания на один существенный момент. С машиной будет взаимодействовать человек, а человеку свойственно ошибаться. Как поведет себя наша программа в ответ на неправильные действия игрока, например, на неверно заданное число – 3? Хорошо продуманная программа фильтрует входные данные; построить строгий фильтр иногда бывает непросто, но делать это нужно, иначе игра может прийти в нерабочее состояние. В приводимой ниже программе SPLOT, написанной на языке Турбопаскаль, такой фильтр предусмотрен.

```
PROGRAM SPLOT (INPUT, OUTPUT) ,
CONST SPORT = 36;
      NUM = 5;
TYPE WID = 1 .. SPORT;
      KART = SET OF WID; .
VAR KLET : WID;
     POSL, UGAD : KART;
     I, J, X, PROBA, R : INTEGER;
(* процедура розыгрыша тиража "Спортлото" *)
PROCEDURE TIRASH (X : INTEGER; VAR POSL : KART);
  VAR I : INTEGER;
       KLET : WID;
  BEGIN I := 0; POSL := [ ];
    REPEAT
      KLET := (RANDOM (X) MOD SPORT) + 1;
      IF NOT (KLET IN POSL)
      THEN BEGIN POSL := POSL + [KLET];
                 I := I + 1;
           END;
    UNTIL I = NUM;
  END;
BEGIN
  WRITELN ( ' Сейчас ЭВМ будет разыгрывать тираж.' );
  WRITELN ( ' Помогите ей, задайте любое целое положительное число ' );
  READ (X);      TIRASH (X, POSL) ,
  WRITELN ( ' А теперь Ваш черед попытать счастья.' );
  WRITELN ( '  Задайте  ', NUM, ' разных целых чисел из интервала 1 .. ',SPORT);
  I := 0; UGAD := [ ];
  FOR J := 1 TO NUM DO
    BEGIN READ (PROBA);
      IF (PROBA >= 1) AND (PROBA <= SPORT)
      THEN IF NOT (PROBA IN UGAD)
           THEN BEGIN UGAD := UGAD + [PROBA];
                  IF PROBA IN POSL
                  THEN I := I + 1
                END
           ELSE
       ELSE WRITELN (PROBA, ' – недопустимое число ' );
     END;
  WRITELN ( ' Вы угадали ', I, ' номеров ' );
  WRITELN ( ' Результаты тиража:   ' );
  FOR J := 1 TO SPORT DO
    IF J IN POSL
    THEN WRITE (J : 4)
END.
```

**Задания**

**А.** Проанализировать работу фильтра от неправильных данных в программе SPLOT.

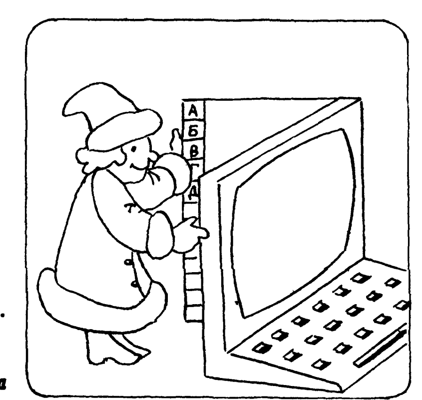
Б. Игрок в "Спортлото 5 из 36" задал 5 чисел. Проверить (не используя множественный тип), нет ли среди них попарно совпадающих или выходящих за заданные границы.

В. Дополнить выполнение программы SPLOT графикой, доступной вашему персональному компьютеру. Например, сначала на экране изображаются 36 пустых прямоугольников. По мере получения случайных чисел прямоугольники переворачиваются или закрашиваются, на них появляется изображение выпавшего числа. Загаданные игроком числа тоже помещаются в соответствующие прямоугольники. Причем это нужно сделать так, чтобы, глядя на экран, сразу можно было понять, какие номера игрок угадал.

## МАЛЕНЬКАЯ ЗАПИСНАЯ КНИЖКА

*Не давай мне ничего на память.*
*Знаю я, как память коротка.*

***А. Ахматова***




"Немного лиц мне память сохранила", – писал А.С. Пушкин. Не надеясь на свою память, в наш век информационного взрыва мы не обходимся без записных книжек. Наша очередная **задача** – создать компьютерную записную книжку. Безусловно, в этой книжке можно хранить самую разнообразную информацию. Чем больше информации, тем более масштабной будет наша задача. Но ее принципиальная сложность от этого не будет особо возрастать, и потому мы начнем создавать маленькую записную книжку, чтобы в основном обратить внимание на все существенные моменты, связанные с задачей подобного рода.

Сначала охарактеризуем информацию, которую должна обрабатывать наша программа, представляющая собой записную книжку. Пусть в книжке – 28 страничек, каждая из которых идентифицируется буквой

русского алфавита от А до Я. На одной страничке можно записать данные о пяти друзьях или знакомых. Для определенности и простоты предположим, что это фамилия, имя и номер телефона. Например, информация на букву С имеет вид:

Серова Марина 32117
Семина Наташа 11789
Сидоров Миша 00000

(Нулевым номером будем обозначать отсутствие телефона).

Таким образом, записная книжка – это массив страничек, на каждой из которых содержится следующая информация:

буква; назовем ее именем BUK;

сведения о человеке (фамилия, имя и телефон); назовем их общим именем ZAP;

количество записей на странице; назовем его именем KOL.

Структура записной книжки приведена ниже (рис. 6).

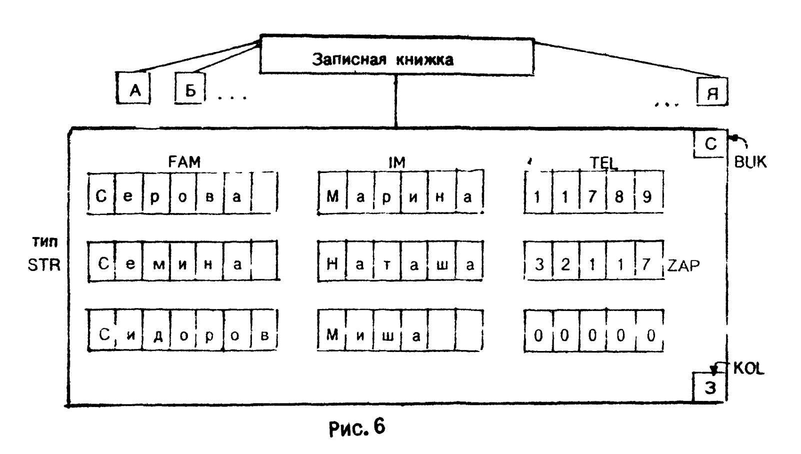



Рис. 6

На языке Паскаль переменная отображающая всю записную книжку, должна быть описана как массив из 28 элементов:

VAR KN: ARRAY [1..28] OF тип элементов.

Обсудим вопрос о типе элементов этого массива. Каждый элемент – сложная структура. В нее входит величина символьного типа (BUK), величина целого типа (KOL) и массив информации о человеке. Информация об одном человеке, в свою очередь, не проста. В одну запись о человеке входят:

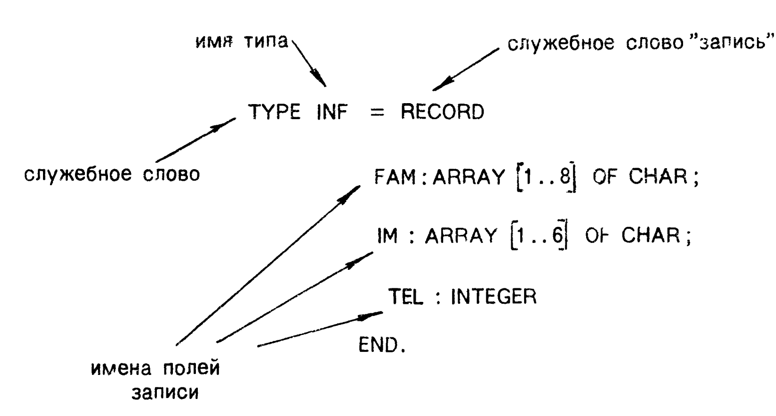
фамилия (FAM) – для определенности предположим, что это массив из 8 символов. Если в фамилии меньше 8 символов, то она дополняется до 8 символом пробела, а друзей с длинными фамилиями (больше 8 символов) у нас нет;

имя (IM) – также для определенности предположим, что это массив из 6 символов;

номер телефона (TEL) – целое число. На первом этапе имеет смысл ограничить его максимально представимым целым числом в вашем компьютере.

Данные характеристики выбраны нами только для определенности и для обеспечения максимальной простоты задачи.

Для удобства работы с подобной информацией (а она встречается весьма часто) в алгоритмических языках определены специальные типы данных, носящие название ЗАПИСИ или СТРУКТУРЫ. Приведем описание типа данных ЗАПИСЬ на языке Паскаль для нашего случая.




Из приведенного описания очевидно важное свойство типа ЗАПИСЬ (RECORD): переменные записи – это объединенные общим именем компоненты разного типа. В нашем примере это массивы символов и целое число. В этом – существенное отличие ЗАПИСИ от МАССИВА, в котором все компоненты должны быть одинаковыми (целый массив, символьный массив, массив массивов, массив записей (!)).

Определив тип ЗАПИСЬ с именем INF, мы можем описать переменную, представляющую одну запись о человеке в нашей записной книжке. Мы вводили для нее имя ZAP. Это переменная типа INF. На одной страничке записной книжки (мы договорились – не более 5 подобных записей, иными словами, массив не более, чем из пяти элементов типа INF. Кроме того, на этой же страничке находится буква (BUK) и целое число (KOL), показывающее, сколько заполненных записей на странице с данной буквой. Таким образом, одна страничка записной книжки это тоже ЗАПИСЬ, но уже с другой структурой.

Назовем эту запись именем STR.

```
TYPE  STR = RECORD
          BUK : CHAR;
          ZAP : ARRAY [1..5] OF INF;
      KOL : INTEGER
   END.
```

Наша записная книжка состоит из 28 компонентов такого типа STR (из 28 страничек). Следовательно, мы можем рассматривать ее в задаче как МАССИВ компонентов типа STR. На языке Паскаль ее описание таково:

VAR KN : ARRAY [1..28] OF STR.

Остается только обратить внимание на то, как просто с помощью типа ЗАПИСЬ (RECORD) объявляется достаточно сложная в иерархическом смысле структура данных.

**Первая задача:** ОФОРМЛЕНИЕ записной книжки.

Она тривиальна по своей сути: на соответствующую страницу нужно внести соответствующую информацию. Что это означает? В компонент BUK на каждой странице должна быть записана соответствующая по порядку буква, а в компонент KOL – занесено нулевое значение, так как записей в книжке пока нет. Предположим, что необходимые буквы находятся в символьном массиве ALF, состоящем из 28 элементов.

Тогда алгоритм в общем виде может быть таким:

алг OFORM
начало алгоритма
 **цикл** по I от 1 до 28
  помещение I-й буквы из массива ALF в компоненту BUK на I-й странице
  занесение нулевого значения в компоненту KOL на I-й странице
 **конец цикла** по I
конец алгоритма OFORM.

Что можно сказать по поводу этого алгоритма?

Первое и второе предложения в теле цикла требуют умения обращаться к отдельным компонентам записной книжки.

Например, надо записать букву A на первую страницу записной книжки, в поле BUK. Естественно, что сначала нужно выделить саму эту страницу. Так как мы имеем дело с массивом страниц, это можно сделать стандартным способом: KN [1]. На этой странице теперь нужно выделить поле BUK, что записывается следующим образом: KN [1]. BUK. Следовательно, KN [2]. BUK – поле BUK на второй странице и KN [I]. BUK – поле BUK на I-й странице. Соответственно – KN [I].KOL – поле KOL на I-странице записной книжки.

Написанные конструкции носят название **уточняющего имени.** Сначала мы выделили страницу, а потом **уточнили**, что нас интересует на этой странице.

Таким образом, первое и второе предложения в теле цикла алгоритма могут быть записаны так:

```
KN [I] . BUK := ALF [I]
KN [I] . KOL := 0.
```

После работы алгоритма OFORM в нашей записной книжке будут представлены постранично все буквы, и она будет пустой (все KOL равны нулю).

**Вторая задача:** ЗАНЕСЕНИЕ (ДОБАВЛЕНИЕ) информации в записную книжку. Мы договорились для определенности, что на одной страничке не может быть больше пяти информационных строчек. Поэтому, прежде чем заносить информацию на страницу с требуемой буквой, надо проверить, не заполнена ли она до конца. Для этого нужно выделить компоненту KOL и проверить ее значение. Если оно равно 5, то алгоритм должен выдать сообщение о том, что занесение невозможно, а иначе он должен поместить информацию на первое свободное место на данной странице. Это место в массиве ZAP определяется значением выражения KOL + 1, которое одновременно определяет и новое значение поля KOL на этой странице.

Алгоритм ЗАНЕСЕНИЯ в общем виде:

<u>алг</u> ZANES
  ВВОД информации для занесения
  выделение первой буквы фамилии
  определение номера (NOM) первой буквы в массиве ALF
  **проверка:** KN [NOM] . KOL = 5?
    ДА: ВЫВОД "Занесение невозможно"
    НЕТ: KN [NOM] . KOL : = KN [NOM] . KOL + 1
      занесение информации в строку KN [NOM] . ZAP [KOL]
  **конец проверки**
<u>конец</u> алгоритма ZANES.

Рассмотрим более детально предложение "Занесение информации в строку". Это предложение означает, что в поле FAM надо поэлементно занести все буквы фамилии, в поле IM – все буквы имени и в поле TEL – номер телефона. Детальная запись этих действий зависит от того, в какой форме будет вводиться информация для занесения. Мы проведем детализацию для случая ввода отдельных букв, чтобы акцентировать внимание на **уточняющем имени.**

Если мы хотим внести информацию о Сидорове Мише в первую информационную строку на страничке с буквой С (ее номер равен 17), то

ЗАНЕСЕНИЕ первой буквы фамилии – KN [17] . ZAP [1] . FAM [1] : = "С";
второй буквы фамилии – KN [17] . ZAP [1] . FAM [2] : = "И";
и т.д.;
первой буквы имени – KN [17] . ZAP [1] . IM [1] : = "М";
и т.д.
номера телефона – KN [17] . ZAP [1] . TEL : = 0, так как у Миши нет телефона.

Из приведенного примера видно, что **уточняющие имена** стали длиннее, так как в данном случае элементы, к которым необходимо обратиться, – массивы (и ZAP, и FAM).

Процесс уточнения имен следующий (рис. 7).

**Третья задача** – ИСПРАВЛЕНИЕ информации.

Здесь появляется новое действие – ПОИСК подлежащей исправлению информации в записной книжке. Например, у Сидорова Миши появился

телефон. Пусть информация для исправления задана в виде значений двух массивов (FAM и IM) и номера телефона (TEL):

СИДОРОВ МИША 30682.

Как найти в записной книжке Сидорова Мишу?

Сначала надо открыть страницу с первой буквой фамилии (т.е. выделить эту букву), потом на этой странице найти запись о Мише. Отметим, что это можно сделать при выбранном способе представления информации в записной книжке (массивы) поэлементным сравнением фамилий и имен. Только найдя запись о Сидорове Мише, можно выполнить действие внесения исправления информации о телефоне – записать новое значение в поле TEL. Таким образом, алгоритм в общем виде очевиден:

```
алг ISPR
   ВВОД информации для исправления
   выделение первой буквы фамилии
   определение номера (NOM) буквы в массиве ALF
   цикл по I от 1 до KN [NOM] . KOL
      проверка: найдена запись о Сидорове Мише?
         ДА:  внести исправления
              выйти из цикла по I
      конец проверки
   конец цикла
конец алгоритма ISPR.
```

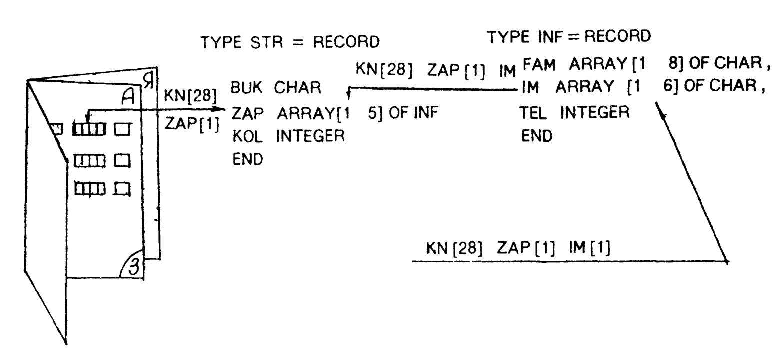



Рис. 7

**Задания**

А. Для двух последних алгоритмов описать возможные способы задания вводимой информации.

Б. Детализировать предложение "Найти номер первой буквы фамилии в массиве ALF".

В. Заданы две фамилии. Написать алгоритм, проверяющий, совпадают ли они.

Ниже для образца приведена маленькая программа на языке Паскаль, которая работает с записной книжкой из 5 страниц.

*ЗАМЕЧАНИЯ*

Если информация из записной книжки хранится, как это предложено в рассмотренной задаче, в оперативной памяти компьютера, то для выполнения любых действий с этой информацией (любых заданий или упражнений) нужно сначала всегда оформлять и заполнять записную книжку. Эти алгоритмы становятся обязательной частью других задач. Если же говорить о реальной ситуации, то желательно постоянно хранить информацию из записной книжки в компьютере (это и была наша цель) и по мере необходимости пользоваться ею. Следовательно, она должна быть записана не в оперативной памяти, а на внешнем носителе информации, например, на гибком диске. Информация, хранящаяся на внешнем носителе в виде некоторой организованной совокупности данных, называется ФАЙЛОМ. В нашем случае содержимое записной книжки должно быть файлом, и тогда все задачи по работе с этой книжкой не будут зависеть друг от друга. Мы не рассматриваем файловой организации данных, но предложенная задача может быть использована для такого рассмотрения.

```
PROGRAM NOTEBOOK (I, O);
CONST N = 5;
TYPE INF = RECORD
             FAM : PACKED ARRAY [1 ..8] OF CHAR;
             IM : PACKED ARRAY[1 .. 6] OF CHAR;
             TEL : INTEGER;
           END;
     STR = RECORD
             BUK : CHAR;
             ZAP : ARRAY [1 .. N] OI INF;
             KOL : INTEGER
           END;
VAR KN . ARRAY[1 .. N] OF STR;
    I, J : INTEGER;
    A : CHAR;
    B : BOOLEAN;
BEGIN (* заполнение записной книжки *)
   KN [1].BUK := 'И'; KN [2].BUK := 'К'; KN [3].BUK := 'П';
   KN [4].BUK := 'С'; KN [5].BUK := 'Т';
   FOR I := 1 TO N DO
      KN [I].KOL := 0;
   KN [1].ZAP [1].FAM := 'ИВАНОВ '; KN [1].ZAP [1].IM := 'ОЛЕГ ';
   KN [1].KOL := KN [1].KOL + 1; KN [3].ZAP [1].FAM := 'ПЕТРОВ ';
   KN [3].ZAP [1].IM := 'ВАНЯ  '; KN [3].KOL := KN [3].KOL + 1;
   KN [3].ZAP [2].FAM := 'ПОПОВА '; KN[3].ZAP[2].IM:=' ГАЛЯ ';
   KN [3].KOL := KN [3].KOL + 1; KN [4].ZAP [1].FAM := 'СЕРОВА ';
   KN [4].ZAP    [1].IM := 'МАРИНА'; KN [4].KOL := KN [4].KOL + 1;
```

```
  KN [4].ZAP [2].FAM:='СЕМИНА'; KN [4].ZAP [2].IM:='НАТАША';
  KN [4].KOL:=KN [4].KOL+1; KN [4].ZAP [3].FAM:='СИДОРОВ ';
  KN [4].ZAP [3].IM:='МИША '; KN [4].KOL:=KN [4].KOL+1;`
  KN [1].ZAP [1].TEL:=23140; KN [3].ZAP [2].TEL:=00000;
  KN [4].ZAP [1].TEL: =11789; KN[4] .ZAP [2].TEL:=32117;
  KN[4].ZAP [3].TEL:=00000;
        (* просмотр записной книжки *)
  WRITELN ('СТРАНИЦУ С КАКОЙ БУКВОЙ ХОТИТЕ ПОСМОТРЕТЬ?);
  B:=TRUE;
  WHILE B DO
     BEGIN
       READLN (A);
       IF (A='С') OR (A='П') OR (A='И') OR (A='К') OR (A='Т')
         THEN
           BEGIN I:=1;
             WHILE A<>KN[I].BUK DO I:=I+1;
             FOR J:=1 TO KN[I].KOL DO
               WRITELN (KN[I].ZAP[J].FAM:12, KN[I].ZAP[J].IM:10,
                        KN[I].ZAP[J].TEL:8);
             B:=FALSE;
           END
  ELSE WRITELN ('ВЫ НЕПРАВИЛЬНО НАБРАЛИ БУКВУ, ПОВТОРИТЕ');
  END
END.
```

*УПРАЖНЕНИЯ*

1. Модифицируйте приведенную программу для своего конкретного случая.
2. Добавьте информацию об одном человеке на заданную букву. Проверьте, возможно ли это добавление.
3. Измените полностью номер телефона у заданного товарища.
4. Измените первую цифру в номере телефона у заданного товарища.
5. Подсчитайте, у скольких из ваших друзей нет телефона (нулевой номер в записной книжке.
6. Определите, на какую букву у вас больше всего друзей.
7. Добавьте в стуктуру записи информацию об адресе (улица, номер дома, номер квартиры).
8. Определите, у кого из друзей номера телефонов – четные.
9. Определите, есть ли среди ваших друзей обладатели "счастливых" номеров телефонов (номер состоит из одинаковых цифр).
10. Определите, у кого из друзей фамилия состоит из четырех букв.
11. Придумайте сами упражнения для данной задачи.

*ЗАМЕЧАНИЕ*

Все упражнения, связанные с номерами телефонов, предполагают умение работать с целыми числами, например, выделять отдельные разряды числа и заменять их. Следовательно, данная задача может служить обоснованием для разработки алгоритмов такого класса. Варианты таких алгоритмов можно найти в задаче "Кросснамбер".

# РАСПИСАНИЕ УРОКОВ

*Созерцание без мышления утомляет. Когда у меня нет новых и новых идей для обработки, я точно больной.*

***В. Гете***

"Составить расписание уроков на один день занятий в восьмом классе, учитывая следующие предварительные сведения.

а) учитель истории может провести либо первый, либо второй, либо третий уроки;

б) учитель литературы может провести либо второй, либо третий урок;

в) математик готов провести либо первый, либо второй урок;

г) преподаватель физкультуры согласен провести только последний урок;

д) в планируемый день занятий у учащихся должно быть четыре разных урока".

С тем, что **ЗАДАЧА** составления расписания актуальна, согласятся, наверное, все. Во всех учебных заведениях, по крайней мере, два раза в год, а в школах – для каждой четверти составляется расписание. Что же касается трудоемкости такого занятия, то этим можно поинтересоваться у завуча вашей школы. Мы думаем, что школьное руководство будет очень радо получить в свое распоряжение программу – составитель расписания.

Когда расписание составляет человек, он продумывает различные варианты, используя и свой предыдущий опыт, и те сведения, которые ему подают учителя, учитывая наличие свободных классов и т.д. Все эти действия носят ярко выраженный логический характер. Более того, величны, которыми приходится манипулировать при решении, тоже имеют нечисловую природу. Что же это за данные? Как можно представить их в алгоритме?

Рассмотрим одно из исходных данных. Пусть это будет фраза: Преподаватель физкультуры согласен провести только четвертый урок". Опыт работы с текстами, которые представляются в памяти ЭВМ в виде строк или массивов символов, у нас имеется. Но, очевидно, в данной ситуации он нам не пригодится. В конце концов, неважно, какими буквами записана фраза, важен ее смысл. И если бы этот смысл был многозначным, то мы вряд ли справились бы с поставленной задачей. Здесь же нам поможет то обстоятельство, что у нашей фразы есть одно интересное свойство: она истинна. А вот фраза: "Преподаватель физкультуры проведет "третий урок" в контексте нашей задачи всегда ложна, учитель физкультуры не согласен ни на первый, ни на второй, ни на третий уроки.

Для того, чтобы основательнее разобраться с данными такой природы, немного отвлечемся от нашей задачи. Дадим некоторые определения, которые помогут нам в дальнейшей работе.

Определение 1. Простое повествовательное предложение, относительно которого можно утверждать, что оно истинно или ложно, называется простым высказыванием.

Примеры высказываний:

1. 73 < 15.
2. В русском алфавите 10 гласных букв.
3. 2 * K + 1 – формула для вычисления нечетных чисел.
4. Костя умеет играть в шахматы.

Простые высказывания 2 и 3 – всегда истинны, 1 – ложно, а 4 – либо истинно, либо ложно в зависимости от Костиных способностей.

А вот следующие фразы не являются высказываниями:

1. Когда ты придешь домой?
2. Сделай доброе дело.

Итак, мы собираемся работать с простыми высказываниями, ибо именно такими фразами выражено условие задачи. Поскольку они могут принимать разные значения, будем считать их значениями переменных. Например, переменная L1 имеет значение ИСТИНА, если первым уроком будет литература, и значение ЛОЖЬ – в противном случае.

Определение 2. Переменные, принимающие значения ИСТИНА или ЛОЖЬ (TRUE или FALSE), называются логическими или булевыми.

Для представления значений таких переменных в памяти ЭВМ достаточно 1 бита: значение = 0 означает ЛОЖЬ, значение = 1 – ИСТИНА. Однако в языках программирования для этих переменных отводится 1 байт или 1 слово, поскольку только эти единицы памяти имеют адреса, по которым можно найти значение. В некоторых языках программирования нет специального описания данных логического типа (Бейсик, Си). Вместо них в Си используются целочисленные переменные со значениями 0 (ЛОЖЬ) или 1 (ИСТИНА). Такой же принцип работы с данными логической природы можно перенести и на Бейсик, если ваш компьютер не "знает" другого языка.

Решив использовать в своих программах булевы переменные, мы

должны узнать, какие операции допустимы над ними в том или ином языке, имеются ли специальные функции, позволяющие упростить работу по составлению алгоритма.

Определение 3. Для описания логической операции используется ТАБЛИЦА ИСТИННОСТИ, в которой указывается результат операции в зависимости от значений операндов. Назовем операнды именами А и В.

Логическое сложение (дизъюнкция) – это операция, позволяющая объединить два простых высказывания в одно сложное, которое будет истинным, если хотя бы одно из простых истинно. В алгоритмическом языке знаком операции логического сложения является служебное слово или, в языках программирования – OR. Составим таблицу истинности логического сложения:

| А | В | А или В |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Используя приведенные выше простые высказывания, запишем пример логического выражения с операцией дизъюнкции: "37 < 15 или Костя умеет играть в шахматы". Поскольку первое высказывание ложно, то значение выражения будет истинно, только если Костя действительно умеет играть в шахматы.

Логическое умножение (конъюнкция) – операция, объединяющая два простых высказывания в одно сложное, которое будет истинным, если оба простых истинны. Знак операции – и (в языках программирования – AND). Таблица истинности операции конъюнкции имеет вид:

| А | В | А и В |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

Примером использования операции конъюнкции может послужить выражение: "37 < 15 и в русском алфавите 10 гласных букв". Это сложное

высказывание всегда ложно, так как ложно первое простое высказывание.

Логическое отрицание (инверсия) – это операция над одним операндом, результат ее – противоположное операнду значение. Знак операции записывается служебным словом не (в языках программирования – NOT). Таблица истинности операции логического отрицания имеет вид:

Старшей среди логических операций является не (NOT), затем следует и (AND), последняя – или (OR).

*УПРАЖНЕНИЯ*

1. В простейших версиях Бейсика нет данных логического типа и не реализованы логические операции. Записать на таком Бейсике действия эквивалентные вычислению выражений с операциями логического сложения, умножения, отрицания, используя в качестве операндов переменные целого типа со значениями 1 и 0. Например, А и В – А * В.
2. Переписать на Бейсик (версия без логических операций) операторы присваивания:

   B := (D или F) и не Q;
   B := (не D и F) или Q;
   B := не (D или F) или Q;
   B := D или F и Q;
   B := не (D или F и Q).

3. Вычислить значение переменной

   W3 := 0 и не N,

   где 0 и N – простые высказывания:
   0 – в нашем классе есть отличники;
   N – в нашем классе есть неуспевающие.

4. Вычислить значение переменной

   W4 := (P1 и P2) или ((P2 или P3) и ИСТИНА)

   при всех возможных сочетаниях значений логических переменных P1, P2, P3. Результат оформить в виде таблицы истинности.
5. В елочной гирлянде последовательно соединены 5 лампочек. Назовем простое высказывание "Лампочка номер I перегорела" – LI. Записать сложное высказывание, которое будет истинным, если гирлянда работает, и ложным – в противном случае.

   Как будет выглядеть аналогичное высказывание при параллельном соединении лампочек?

Законы преобразования сложных логических выражений составляют предмет БУЛЕВОЙ АЛГЕБРЫ. Интересующимся рекомендуем познакомиться с литературой [22].

**Задание** А. Придумать наглядную иллюстрацию логических операций.

Вернемся теперь к ЗАДАЧЕ о расписании и рассмотрим ее с учетом новых знаний.

## РАЗРАБОТКА АЛГОРИТМА

Исходные данные задачи – простые высказывания, они могут принять значения ИСТИНА или ЛОЖЬ. Будем считать каждое высказывание значением отдельной логической переменной, поименуем эти переменные:

I1 (I2, I3) – учитель истории проводит первый (второй, третий) урок;

L2 (L3) – учитель литературы проводит второй (третий) урок;

M1 (M2) – математик проводит первый (второй) урок;

F4 – преподаватель физкультуры проводит четвертый урок.

В нашем распоряжении 8 логических операндов, с их помощью мы должны записать условие: расписание составлено верно. Очевидно, этим условием будет логическое выражение, структура которого отобразит взаимосвязь между исходными простыми высказываниями. Значение формируемого выражения присвоим переменной DEN (расписание на день занятий) и будем отыскивать такие совокупности значений I1, I2,... F4, которые приводят к DEN = ИСТИНА.

Мы знаем, что в день у учеников должно быть четыре разных урока. Условие того, что один урок истории состоялся, запишется в виде некоторого логического выражения, его значение присвоим переменной IST. Аналогично для других предметов будем использовать переменные LIT, MAT и FIZ. Тогда о правильности расписания будут говорить истинные значения этих переменных. Кроме того, в правильно составленном расписании не должно быть наложений, например, не могут быть одновременно первым уроком история и математика. Обозначим условия отсутствия наложений в расписании UR1 (урок № 1 – либо история, либо математика), UR2, UR3. Для четвертого урока такое условие можно не записывать, так как кроме преподавателя физкультуры на это время никто не претендует, т.е. условие проведения урока № 4 уже указывает переменная FIZ. Когда все условия IST, LIT, MAT, FIZ, UR1, UR2 и UR3 одновременно станут истинными, должна быть истинна и DEN. Значит, можно записать

DEN : = IST и LIT и MAT и FIZ и UR1 и UR2 и UR3.

А теперь последовательно выведем зависимость каждого отдельного условия от исходных данных. Начнем с урока истории. Условие того, что учитель истории провел именно первый урок:

I1 и не I2 и не I3.

Аналогично запишутся условия для второго и третьего уроков. Какой-то урок все же должен состояться, поэтому

IST : = (I1 и не I2 и не I3) или
(не I1 и I2 и не I3) или
(не I1 и не I2 и I3).

Подобным образом можно записать выражения для LIT и MAT, а FIZ всегда равно F4 и всегда истинно, но для полноты записи мы оставим это условие среди прочих.

Если говорить об уроке с определенным номером, то тогда условия будут выглядеть так:

UR1 : = (I1 и не M1) или
(не I1 и M1),

аналогично для UR2 и UR3.

*Упражнение* 6. Записать в виде логического выражения фразу: "В соревновании победит Света или Наташа, а Катя будет вотоpой".

Итак, мы связали друг с другом в логические выражения все исходные данные задачи. Теперь предстоит определить, для какой совокупности значений переменных I1, I2, I3, L2, L3, M1, M2, F4 условие DEN будет истинным, т.е. в предположении, что DEN – некоторая функция указанных переменных, нужно построить таблицу истинности для полученной функции и выбрать из нее строки, где DEN = ИСТИНА. Таблица истинности для 8 независимых переменных будет содержать $2^8 = 256$ строк.

**Задание** Б. Доказать, что таблица истинности функции $m$ переменных содержит $2^m$ строк.

Так ход решения привел нас к самостоятельной задаче: создать логический массив размером 256★8, в котором ни одна строка не повторяется. Для наших целей хранить в памяти ЭВМ весь массив не нужно. Достаточно получить одну строку, подсчитать значение DEN, если DEN = ИСТИНА, напечатать эту строку как очередной результат и перейти к созданию новой строки. Но выполнять такие действия нужно 256 раз. Ввиду вспомогательного значения этого алгоритма. с одной стороны, а с другой – потребности многократного обращения к нему его нужно оформить в виде процедуры с выходным параметром – массивом логического типа, длиной 8. Без массива невозможно организовать циклические действия, а у нас данные – 8 простых переменных. Значит, нужно поставить в соответствие каждой из этих переменных определенный элемент массива, например,

| элемент массива | A[1] | A[2] | A[3] | A[4] | A[5] | A[6] | A[7] | A[8] |
|---|---|---|---|---|---|---|---|---|
| логическая переменная | I1 | I2 | I3 | L2 | L3 | M1 | M2 | F4 |
| значение | TRUE | FALSE | FALSE | FALSE | TRUE | FALSE | TRUE | TRUE |

Если данные этой таблицы подставить в условие DEN, то увидим, что оно будет истинным. Следовательно, мы записали пример возможного расписания:

1. История
2. Математика
3. Литература
4. Физкультура.

Общая структура алгоритма будет следующая:

алг RASPIS
описания данных
начало алгоритма

первая строка таблицы истинности – все значения ЛОЖЬ;
<u>для</u> К <u>от</u> 1 <u>до</u> 256
инициализация переменных I1, I2, I3, L2, L3, M1, M2, F4
значениями из таблицы истинности;
вычисление значений переменных IST, LIT, MAT, FIZ, UR1, UR2, UR3, UR4, DEN;
<u>если</u> DEN
<u>то</u> напечатать очередной результат
<u>все</u>
получение следующей строки таблицы истинности
<u>конец</u> <u>цикла</u> по К
<u>конец</u> алгоритма RASPIS.

В этом алгоритме использован условный оператор с условием – логической переменной DEN, которая представляет собой частный случай логического выражения. Ниже мы встретимся с ситуацией, когда логическая переменная используется в качестве условия в цикле <u>пока</u>, как это сделано, например, в задаче "Мухи, слоны и числа".

Все приведенные действия достаточно просты, за исключением формирования строк таблицы истинности. Как это часто бывает при детализации алгоритмов, нам снова предстоит здесь решить самостоятельную задачу, которая может оказаться полезной при составлении других программ.

Чтобы получать строки последовательно одну за другой без повторений, можно, например, использовать аналогию с правилами двоичного сложения. Представим мысленно, что строка – это некое двоичное число, в котором 0 соответствует значению ЛОЖЬ, а 1 – ИСТИНА. Чтобы получить следующее по величине число, к исходному нужно прибавить 1. Выполним это сложение в двоичной системе счисления. Вспомним правила двоичного сложения:

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$ (1 – перенос в соседний разряд).

Например,

$$\begin{array}{r} 0\,1\,0\,0\,1\,0\,1\,1 \\ + \qquad\qquad 1 \\ \hline 0\,1\,0\,0\,1\,1\,0\,0 \end{array}$$

Замечаем, что только сложение двух значений 1 вынуждает нас перейти в старший разряд и продолжить действия. А в тех случаях, когда перенос в соседний разряд равен 0, значения в старших разрядах не меняются, и вычисления можно прекратить. Иначе можно сказать, что для получения следующего числа нужно изменить значения в разрядах исходного числа на противоположные, начиная с младшего разряда и до первого нулевого значения включительно. Этот алгоритм можно использовать

для получения строк таблицы истинности. Например, по аналогии с предыдущим примером вслед за строкой

ЛОЖЬ ИСТИНА ЛОЖЬ ЛОЖЬ ИСТИНА ЛОЖЬ ИСТИНА ИСТИНА

должна генерироваться строка

ЛОЖЬ ИСТИНА ЛОЖЬ ЛОЖЬ ИСТИНА ИСТИНА ЛОЖЬ ЛОЖЬ

Эти действия можно оформить с помощью цикла типа пока с досрочным выходом при обнаружении элемента со значением ЛОЖЬ. Для организации этого выхода будем использовать логическую переменную PERENOS (по аналогии с переносом в двоичном сложении). Кроме значения переменной PERENOS условием работы этого цикла является ограничение по номеру разряда, иначе мы рискуем выйти за левый край последовательности. Возьмем обычный целочисленный счетчик, который сработает только для значения K = 256.

**Задание** В. Доказать, что ограничение в цикле по номеру разряда срабатывает только для K = 256.

Получаем довольно простой алгоритм создания очередной строки таблицы истинности для функции восьми переменных:

```
алг ISTTAB (лог таб A[1:8])
  арг A
  рез A
  нач лог PERENOS
      цел I
      PERENOS := ИСТИНА
      I := 8
      пока PERENOS и I > 0
         если A[I]
            то A[I] := ЛОЖЬ
            иначе A[I] := ИСТИНА
                  PERENOS := ЛОЖЬ
         все
         I := I - 1
       конец цикла пока
 конец алгоритма ISTTAB.
```

Отмечаем еще раз своеобразие применения логических переменных. PERENOS – это уже само по себе условие, так как его значение ИСТИНА или ЛОЖЬ. A [I], с одной стороны, обычный элемент массива, а с другой – условие для условного оператора.

Задача о расписании взята из [21], где она решена с использованием графов. Текст программы RASPIS на языке Паскаль приведен ниже.

```
PROGRAM RASPIS (INPUT, OUTPUT),
CONST N = 8 ,
TYPE MAS = ARRAY [1 . N] OF BOOLEAN ;
```

```
VAR A : MAS ;
     K, I, J : INTEGER ;
     I1, I2, I3, L2, L3, M1, M2, F4 : BOOLEAN ;
     IST, LIT, MAT, FIZ, UR1, UR2, UR3, DEN : BOOLEAN ;
(* процедура генерации очередной строки таблицы истинности *)
PROCEDURE ISTTAB (N : INTEGER ; VAR A : MAS) ;
  VAR I : INTEGER ;
      PERENOS : BOOLEAN ;
  BEGIN PERENOS : = TRUE ; I : = N ,
    WHILE PERENOS AND (I > 0) DO
      BEGIN IF A [I]
              THEN A [I] : = FALSE
              ELSE BEGIN A [I] : = TRUE ; PERENOS : = FALSE
                   END ;
              I : = I - 1
      END
  END ;


(* функция вычисления A в степени N *)
FUNCTION STEP (A, N : INTEGER) : INTEGER ;
  VAR REZ, POK, R : INTEGER ;
  BEGIN REZ : = 1 ; K : = A ; POK : = N ;
    WHILE POK > 0 DO
      BEGIN IF (POK MOD 2) <> 0
              THEN BEGIN POK : = POK - 1 ;
                         REZ : = REZ * K
                   END ;
              IF POK > 0
              THEN BEGIN POK : = POK DIV 2 ;
                         K : = K*K
                   END
      END ;
    STEP : = REZ
  END ;
BEGIN
  FOR I : = 1 TO N DO A [I] : = FALSE ;

 (* определение размера таблицы истинности *)
  K : = STEP (2, N) ;
  FOR J : = 1 TO K DO
    BEGIN
      I1 : = A [1] ; I2 : = A [2] ; I3 : = A [3] ; L2 : = A [4] ; L3 : = A [5] ;
      M1 : = A [6] ; M2 : = A [7] ; F4 : = A [8] ;
      IST : = (I1 AND NOT I2 AND NOT I3) OR
             (NOT I1 AND I2 AND NOT I3) OR
             (NOT I1 AND NOT I2 AND I3) ;
      LIT : = (L2 AND NOT L3) OR (NOT L2 AND L3) ;
      MAT : = (M1 AND NOT M2) OR (NOT M1 AND M2) ;
      FIZ : = F4 ;
```

```
      UR1 := (I1 AND NOT M1) OR (NOT I1 AND M1);
      UR2 := (I2 AND NOT L2 AND NOT M2) OR
             (NOT I2 AND L2 AND NOT M2) OR
             (NOT I2 AND NOT L2 AND M2);
      UR3 := (I3 AND NOT L3) OR (NOT I3 AND L3);
      DEN := IST AND LIT AND MAT AND FIZ AND UR1 AND UR2 AND UR3;
      IF  DEN
      THEN BEGIN WRITELN; (* печать варианта расписания *)
                 FOR I := 1 TO N DO WRITE (A [I] : 6);
           END;
      ISTTAB (N, A)
   END
END.
```

Предложенный вариант решения задачи оказывается чувствительным к изменениям совокупности исходных данных. Например, если в учебном дне школьника появится еще один, пятый урок, то, следуя принятой логике рассуждений, придется вводить несколько новых переменных – условий проведения этого урока. Тогда все выражения в тексте программы изменятся, т.е. нужно будет составлять новый алгоритм.

Рассмотрим другой способ представления исходной информации, который приведет в дальнейшем к некоторому достаточно универсальному алгоритму. Все требования, предъявляемые разными учителями к расписанию уроков, представим в виде таблицы:

| Номер урока | IST | LIN | MAT | FIS |
|---|---|---|---|---|
| 1 | ИСТИНА | ЛОЖЬ | ИСТИНА | ЛОЖЬ |
| 2 | ИСТИНА | ИСТИНА | ИСТИНА | ЛОЖЬ |
| 3 | ИСТИНА | ИСТИНА | ЛОЖЬ | ЛОЖЬ |
| 4 | ЛОЖЬ | ЛОЖЬ | ЛОЖЬ | ИСТИНА |

Значение ИСТИНА здесь означает, что в соответствующее время может состояться указанный урок, ЛОЖЬ – урока быть не может. Поскольку наложений и сдвоенных уроков в расписании быть не должно, наша **ЗАДАЧА** – построить таблицы, подобные приведенной выше, в которых значение ИСТИНА встречается только 1 раз в каждой строке и в каждом столбце, в то же время местоположение этого значения согласуется с исходными ограничениями. Например, правильно составленное расписание:

| Номер урока | IST | LIT | MAT | FIS |
|---|---|---|---|---|
| 1 | ИСТИНА | ЛОЖЬ | ЛОЖЬ | ЛОЖЬ |
| 2 | ЛОЖЬ | ЛОЖЬ | ИСТИНА | ЛОЖЬ |
| 3 | ЛОЖЬ | ИСТИНА | ЛОЖЬ | ЛОЖЬ |
| 4 | ЛОЖЬ | ЛОЖЬ | ЛОЖЬ | ИСТИНА |

Неправильное расписание (математика не может быть третьим уроком):

| Номер урока | IST | LIT | MAT | FIS |
|---|---|---|---|---|
| 1 | ИСТИНА | ЛОЖЬ | ЛОЖЬ | ЛОЖЬ |
| 2 | ЛОЖЬ | ИСТИНА | ЛОЖЬ | ЛОЖЬ |
| 3 | ЛОЖЬ | ЛОЖЬ | ИСТИНА | ЛОЖЬ |
| 4 | ЛОЖЬ | ЛОЖЬ | ЛОЖЬ | ИСТИНА |

Такая интерпретация исходной информации оказывается удачной вследствие того, что 1) изменение количества входных ограничений (условий проведения уроков или добавление новых уроков) не изменяет структуры данных, меняются только размеры таблицы; 2) изменение количества входных ограничений никак не влияет на алгоритм.

В качестве самостоятельной работы предлагается реализовать этот более совершенный способ решения задачи.

**Задания**

Г. Переделать алгоритм RASPIS с учетом того, что всегда F4 = ИСТИНА.
Д. Придумать другой алгоритм создания таблицы истинности.
Е. Дополнить программу RASPIS красивой печатью результатов.
Ж. Решить с помощью ЭВМ задачу о соревнованиях [21]. В школе проводятся соревнования по плаванию. Болельщики высказывают следующие предположения о будущих победителях:

Ваня: "Наташа будет первой, а Вера – второй".
Сережа: "Первой будет Света, а Люда займет третье место".

Дима : "Света будет второй, Вера может рассчитывать лишь на третье место".

По окончании соревнований выяснилось, что каждый из них в одном из двух своих предположений оказался прав.

Кто из участниц занял первое, второе и третье место, если известно, что каждое место заняла одна из них и все они были призерами?

3. Разобраться с помощью ЭВМ в спорной ситуации [33].

В квартире раздался звон разбитого стекла, и в окно влетел футбольный мяч. Хозяин выбежал с мячом в руках во двор и увидел девятерых растерянных мальчишек.

– Кто это сделал? – грозно спросил он.

Андрей: "Это сделал Дима".

Борис: "Это неправда".

Виктор: "Окно разбил я".

Геннадий: "Это сделал Виктор или Иван. Борис говорит неправду".

Евгений: "Разбил Виктор".

Егор: "Нет, Виктор не разбивал".

Иван: "Ни Виктор, ни я здесь не при чем".

Константин: "Иван прав, но Дмитрий тоже не виноват".

Позже выяснилось, что трое мальчиков говорили правду. Кто же разбил стекло?

В заключение отметим, что существуют специальные языки программирования для работы с данными логической природы, например, Ляпас [18]. В нем так же, как в Си, отдельному логическому значению отводится 1 бит памяти и в зависимости от размера слова памяти (16, 32 или 64 бита) логические операции выполняются сразу над группой данных (16, 32 или 64 элемента), что заметно сокращает время работы программы в ЭВМ.

# ПРИЛОЖЕНИЕ

## РАБОТА С ЛИСТАМИ ОПОРНЫХ СИГНАЛОВ

Листы опорных сигналов (ЛОС) – один из наиболее известных и употребляемых элементов организационно-методической системы В.Ф. Шаталова. Их применение при изучении курса информатики, безусловно, должно основываться на специфике этого курса. Для рассматриваемого в настоящее время варианта содержания курса эта специфика связана, в частности, с тем, что теоретический компонент его является незначительным по объему. Главная задача курса чаще всего формулируется как привитие практических навыков и умений использовать возможности компьютера. Это положение и определяет характер построения и использования листов в учебном процессе.

ИСКУССТВО ЗАПОМИНАНИЯ

**Дезидерий, Эразмий**

Д. Как продвигаются твои дела, Эразмий ?

Э. Кажется, Музы не очень ко мне благосклонны. Но дело пошло бы удачнее, если бы я мог кое-что от тебя получить.

Д. Отказа ни в чем не встретишь – только бы это было тебе на пользу. Итак, говори.

Э. Не сомневаюсь, что нет ни единого из тайных искусств, которого бы ты не знал.

Д. Если бы так!

Э. Говорят, существует некое искусство запоминания, которое позволяет почти без хлопот выучить все свободные науки.

Д. Что я слышу! И ты сам видел книгу ?

1. На листах фиксируется в символической, компактной, обобщенной и систематизированной форме учебный материал, как правило, его крупные дозы. В тех вариантах, что предложены в книге, очень невелик процент ассоциативных элементов; в основном это общепринятые схемы и символика.

2. Порядок использования листов может быть таким

а) на этапе первичного ознакомления с новым учебным материалом. В этом случае функция листов – информирующая, и потому большое значение приобретает их внешняя наглядность. Запоминания листов не требуется;

---

Э. Видел. Но именно что видел: учителя не нашлось.

Д. А в книге что ?

Э. Изображения разных животных – драконов, львов, леопардов, разные круги, а в них слова – и греческие, и латинские, и еврейские, и даже из варварских языков.

Д. А в названии указывалось, за сколько дней можно постигнуть науки ?

Э. Да, за четырнадцать.

Д. Щедрое обещание, ничего не скажешь. Но знаешь ли ты хоть одного человека, которого бы это искусство запоминания сделало ученым ?

Э. Нет, ни одного.

Д. И никто иной такого человека не видел и не увидит, разве что сперва мы увидим счастливца, которого алхимия сделала богатым.

Э. А мне бы так хотелось, чтобы это было правдой !

Д. Наверное, потому, что досадно покупать знания ценою стольких трудов.

б) на этапе решения задач и выполнения упражнений. Здесь листы выполняют роль активного справочного материала. Они постоянно "на глазах" у учащихся, и педагог обязан осуществлять обращение к ним при появлении подходящей ситуации в процессе решения задач;

---

Э. Конечно.

Д. Но так судили вышние боги. Обыкновенные богатства – золото, самоцветы, серебро, дворцы, царства – они иной раз дарят и ленивым и недостойным; но истинные богатства, которые доподлинно составляют нашу собственность, нельзя приобрести иначе, как трудом. И нам не должен быть в тягость труд, которым приобретается такая ценность, когда мы наблюдаем, как очень многие сквозь ужасающие опасности и неисчислимые преграды рвутся к благам преходящим и, по сравнению с ученостью, право же, ничтожным, да еще и не всегда достигают цели. А кроме того, к трудам учения примешана и немалая слабость, если подвигаться вперед постепенно. И наконец, ты сам способен утишить и скуку и досаду – в значительной степени это зависит от тебя.

Э. Как так?

Д. Убеди себя, во-первых, полюбить занятия. А во-вторых, – восхищайся ими.

Э. А какие к этому средства?

Д. Задумайся над тем, скольких людей обогатили науки, скольких возвели на вершину почета и власти. И еще задумайся, как велико различие меж человеком и скотом.

Э. Добрый совет.

в) на заключительном этапе (который может быть значительно отнесен во времени от первого) листы целенаправленно используются для организации теоретического обобщения. Здесь предполагается большая работа по освоению всех компонентов листа и их запоминанию. Последнее, как показывает опыт, в данном случае приобретает непроизвольный характер, так как связано с постоянным предшествующим использованием листа в процессе практической работы. Для фиксации усвоения теоретических, обощенных элементов учебного материала к листам прилагаются совокупности вопросов.

---

Д. Далее, нужно приучить ум к сосредоточенности; он должен находить удовольствие прежде в полезном, а не в приятном. То, что само по себе и достойно и славно, поначалу иной раз бывает сопряжено с некоторою тягостью, которая, однако же, рассеивается привычкою. Тогда ты и учителя будешь меньше утомлять, и сам усваивать будешь легче – в согласии со словами Исократа, которые следует начертать золотыми буквами на титульном листе твоей книги: Если будешь любознателен, узнаешь много.

Э. Усваиваю-то я довольно быстро, но все мигом утекает!
Д. Как из дырявой бочки?
Э. Да, именно. А помочь ничем не могу.
Д. Отчего же, надо заделать щели и остановить течь.
Э. Чем заделать?

К приводимым ниже листам авторы предлагают относиться как к возможным вариантам, безусловно, носящим в какой-то мере субъективный характер. Читатели вправе заменять и отдельные части листов, и все структуры целиком, если они найдут более удачный, с их точки зрения, вариант. Тот, кто пожелает использовать листы для организации учебного процесса, может найти и свой, отличный от нашего способ включения их в систему учебных заданий.

---

Д. Не мхом и не гипсом, а усердием. Кто вытверживает слова, не поняв их смысла, скоро их забывает, ибо слова, как говорит Гомер, крылаты и легко улетают, если их не удерживает груз смысла. А стало быть, в первую очередь, старайся понять существо дела, потом обдумай еще и еще раз. И надо, как я уже сказал, приучить ум к тому, чтобы он мыслил сосредоточенно всякий раз, когда потребуется. А если у кого ум до того дикий, что к этому привыкнуть не способен, он для науки решительно не годится.

Э. Я слишком понимаю, как это трудно.

Д. А у кого ум до того верткий, что не может задержаться ни на какой мысли, тот ни слушать долго не может, ни закрепить в памяти то, что узнал. Надежно оттиснуть чтобы то ни было можно на свинце; на воде и ртути, которые всегда струятся и растекаются, нельзя оттиснуть ничего... Если бы тебе удалось приучить к этому свой разум, ты с наименьшими трудами запомнишь очень многое, – ведь ты постоянно находишься среди ученых, чьи беседы каждый день приносят столько достопамятного.

Э. Да, конечно.

Целесообразно закончить этот короткий раздел словами психолога О.К. Тихомирова: "Психологи, работающие в области педагогической психологии, иногда склонны недооценивать значение житейских понятий... В научном, техническом творчесте эмпирические обобщения могут быть теми источниками, "питательным материалом", из которого вырастают первичные гипотезы. Если "отсечь" этот кажущийся несовершенным уровень мышления (т.е. эмпирические обобщения), то можно погубить самое важное звено в процессе мышления – формирование новых гипотез. Необходимо проводить линию на максимальное использование, на органическое сочетание житейских и научных понятий, а не пытаться заменить одно другим".

---

Д. Ведь помимо речей за столом, помимо повседневных разговоров, ты сразу после завтрака выслушиваешь восемь изящных, остро отточенных изречений, выбранных из лучших авторов, и после обеда столько же. Подсчитай-ка, сколько наберется за месяц и за год.

Э. Целая гора, если бы все упомнить.

Д. А раз вокруг говорят только по латыни, и говорят хорошо, что мешает тебе в течение нескольких месяцев выучиться этому языку, когда неграмотные мальчишки за короткий срок выучиваются по-французски или по-испански?

Э. Последую твоему совету и испытаю свой ум – способен ли он привыкнуть к ярму Муз.

Д. Иного искусства запоминания, кроме старательности, любви и усердия я не знаю.

**Эразм Роттердамский (1469–1536),**

"Разговоры запросто"

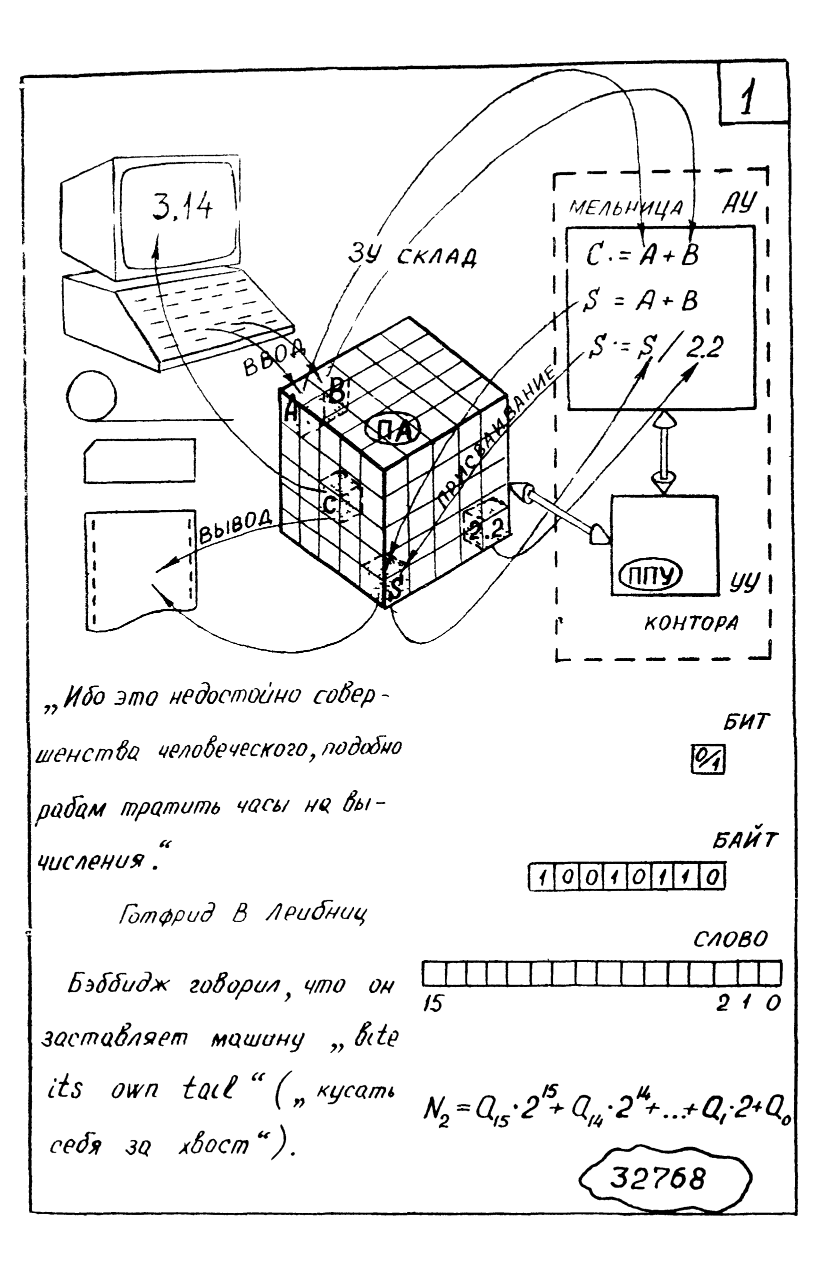
## ПОЯСНЕНИЯ К ЛОС № 1

На схеме, представленной на ЛОС № 1 (здесь и далее номера листов опорных сигналов помечены цифрой в правом верхнем углу), выделены основные аппаратурные блоки современной ЭВМ – память (ЗУ – запоминающее устройство), устройства ввода-вывода (УВВ) и центральный процессор, который для удобства проведения исторической аналогии представлен в виде двух блоков. Это – арифметическое устройство (АУ) и устройство управления (УУ).

В начале XIX в. английским инженером Чарльзом Бэббиджем была выдвинута идея построения счетной машины с ПРОГРАММНЫМ управлением. Ученица Ч. Бэббиджа, дочь английского поэта Дж. Байрона леди Лавлейс, известная сейчас как первая программистка, писала, что машину Бэббиджа "нельзя смешивать с простыми счетными машинами. Она занимает совершенно особе место... среди механизмов, представляющих возможности комбинировать произвольные символы".

Функциональная схема машины Бэббиджа содержала следующие блоки – СКЛАД, МЕЛЬНИЦУ и КОНТОРУ. Эти образные названия приведены на ЛОС и, думается, не нуждается в особых пояснениях тот факт, что они полностью соответствуют определенным функциональным блокам современной ЭВМ.

ПАМЯТЬ ЭВМ на листе опорных сигналов представлена в виде куба, разделенного на отдельные элементарные ячейки – места хранения единиц информации, обработкой которой занимается ЭВМ. Подобное деление кроме того, что оно связано с понятием БАЙТА, а также более и менее крупных ячеек памяти (бит, килобайт, мегабайт и т.д.), позволяет просто объяснить один из основных принципов, положенных в основу современных вычислительных машин и сформулированных крупнейшим математиком нашего века Джоном фон Нейманом. Это принцип АДРЕСАЦИИ ячеек памяти (опорный сигнал – ПА).


1
3.14
ЗУ СКЛАД
МЕЛЬНИЦА
АУ
C := A + B
S = A + B
S := S / 2.2
ВВОД
A
B
ПА
C
ПРИСВАИВАНИЕ
2.2
ВЫВОД
S
ППУ
УУ
КОНТОРА
„Ибо это недостойно совершенства человеческого, подобно рабам тратить часы на вычисления."
Готфрид В Лейбниц
БИТ
0/1
БАЙТ
1 0 0 1 0 1 1 0
СЛОВО
15
2 1 0
Бэббидж говорил, что он заставляет машину „bite its own tail" („кусать себя за хвост").
$N_2 = Q_{15} \cdot 2^{15} + Q_{14} \cdot 2^{14} + \ldots + Q_1 \cdot 2 + Q_0$
32768

Память ЭВМ разбита на элементарные ячейки, которые пронумерованы ПОДРЯД. НОМЕР ячейки есть ее АДРЕС. ЭВМ размещает и разыскивает информацию в памяти по адресам. Таким образом, каждая ячейка характеризуется своим АДРЕСОМ и СОДЕРЖИМЫМ (тем, что в ней находится). Собственно, работа ЭВМ и заключается в том, что она в соответствии с некоторой совокупностью команд ИМЕЕТ СОДЕРЖИМОЕ ЯЧЕЕК ПАМЯТИ.

ПРИНЦИП ПРОГРАММНОГО УПРАВЛЕНИЯ (ППУ) – второй основной принцип, положенный в основу современной вычислительной машины. Он заключается в том, что ЭВМ работает сама без участия человека по программе (последовательности команд), которая находится в ее памяти, т.е. предварительно вводится в нее с какого-то внешнего носителя. ЭВМ выбирает команды из памяти последовательно одну за другой, анализирует их в арифметическом устройстве, выполняет и результаты снова возвращает в запоминающее устройство. И всем этим процессом управляет устройство управления. Именно такой способ работы имел в виду Ч. Бэббидж, когда сказал, что его машина все время "кусает себя за хвост".

В квадрате, соответствующем арифметическому устройству, помещена последовательность трех команд – три оператора (так они называются в алгоритмических языках) ПРИСВАИВАНИЯ ЗНАЧЕНИЙ переменным C и S. При записи операторов использован знак операции присваивания $:=$. После выполнения операторов $C := A + B$ и $S := A + B$ переменные C и S получат значение, равное сумме значений переменных A и B, которые были введены в память ЭВМ с помощью команд ВВОДА. После выполнения операторов $S := S/2.2$ переменная S получит новое значение, которое равно старому значению этой переменной, разделенному на константу 2.2. Последняя тоже берется из некоторой ячейки памяти.

Остановимся еще на одном моменте. Команды присваивания $S := A + B$ и $S := S/2.2$ находились в памяти ЭВМ и были выбраны в арифметическое устройство последовательно друг за другом. После выполнения первой команды в ячейку памяти для переменной S занеслось значение, равное сумме значений переменных A и B, а после выполнения второй команды – значение, в 2.2 раза меньшее. Этот второй оператор присваивания требует особого внимания при объяснении, так как для его выполнения переменная, получающая значение и стоящая слева от знака присваивания, и переменная, значение которой используется при вычислении выражения, записанного справа от знака присваивания, находятся в одной и той же ячейке памяти.

Таким образом, вынесенные на ЛОС команды присваивания помогают сделать два важных дела:

1) пояснить принцип программного управления;

2) пояснить смысл важнейшей операции для написания алгоритмов – операции присваивания. Она легче воспринимается на этом этапе, так как понятия ПЕРЕМЕННОЙ, ВЫРАЖЕНИЯ и ЗНАЧЕНИЯ выражения хорошо знакомы учащимся из курса алгебры.

И еще одно существенное замечание. На ЛОС № 1 выделены два вида операций (команд), которые позволяют менять содержимое ячеек памяти ЭВМ – ВВОД и ПРИСВАИВАНИЕ. В чем разница между ними? При ВВОДЕ

информация вводится с внешнего носителя (извне). При ПРИСВАИВАНИИ информация (новое значение) создается при вычислении выражений с использованием значений внутри ЭВМ. Опорные сигналы, связанные с ВЫВОДОМ информации, не требуют дополнительных комментариев.

## ВОПРОСЫ К ЛОС № 1

1. Какие вопросы изучает наука ИНФОРМАТИКА ?
2. Дайте интуитивное определение понятия АЛГОРИТМ.
3. Из каких функциональных блоков состоит ЭВМ ?
4. Назначение функциональных блоков ЭВМ.
5. Что такое ЦЕНТРАЛЬНЫЙ ПРОЦЕССОР ?
6. Что значит принцип АДРЕСАЦИИ ФОН НЕЙМАНА ?
7. В каком виде представляется информация в ЭВМ ?
8. Что такое БИТ и БАЙТ ?
9. Что хранится в одной ячейке памяти ЭВМ ?
10. Что значит принцип ПРОГРАММНОГО УПРАВЛЕНИЯ ?
11. Где хранится программа, по которой работает ЭВМ ?
12. Каково назначение операторов ВВОДА и ПРИСВАИВАНИЯ ?
13. В чем разница между этими операторами ?

## ПОЯСНЕНИЯ К ЛОС № 2

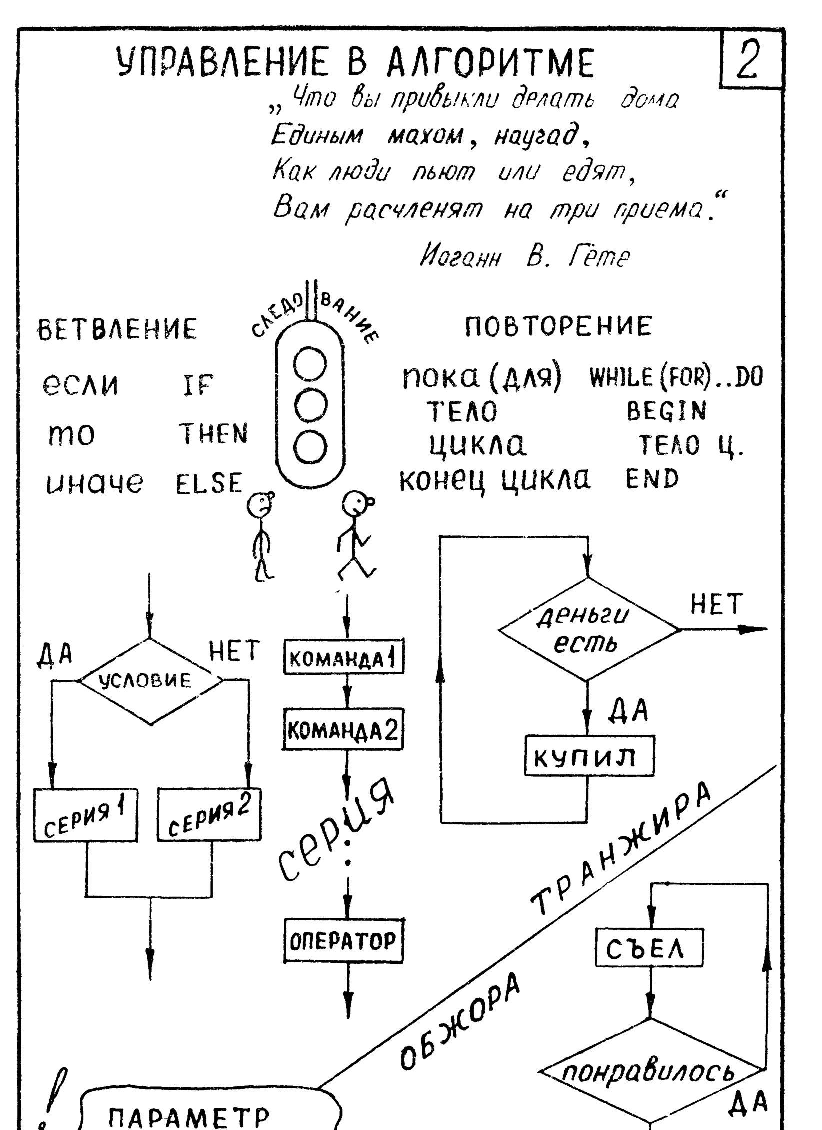
На ЛОС № 2 представлена информация об основных управляющих конструкциях, используемых для записи алгоритмов при структурном подходе. Это конструкция следования (последовательное выполнение команд, написанных подряд, – серии команд), ветвления (выбор пути работы алгоритма в зависимости от результата проверки некоторого условия) и повторения (многократное выполнение серии команд, составляющей тело конструкции повторения – тело цикла). Для всех конструкций приведены служебные слова в русском варианте и используемые в языке Паскаль.

Три названные управляющие конструкции ассоциативно связываются с опорным сигналом в виде светофора. Светофор – управляющий элемент, и динамика его работы хорошо интерпретирует все три команды управления.

На ЛОС № 2 приведены семантические блок-схемы для описанных конструкций. Для команды повторения эти блок-схемы даны в конкретных вариантах. Цикл с проверкой условия перед выполнением тела цикла (с предусловием) представлен как "транжира", а цикл с проверкой условия после выполнения тела цикла (с пост-условием) – как "обжора". Эти две ассоциативные опоры предложены учителем информатики из г. Тбилиси М. А. Шовманом.

## ВОПРОСЫ К ЛОС № 2

1. Может ли команда повторения быть серией в команде ветвления?
2. Составьте общую семантическую блок-схему для команд повторения.
3. Может ли команда ветвления находиться в теле цикла ?
4. Чем принципиально отличается команда повторения типа пока от команды типа для ?

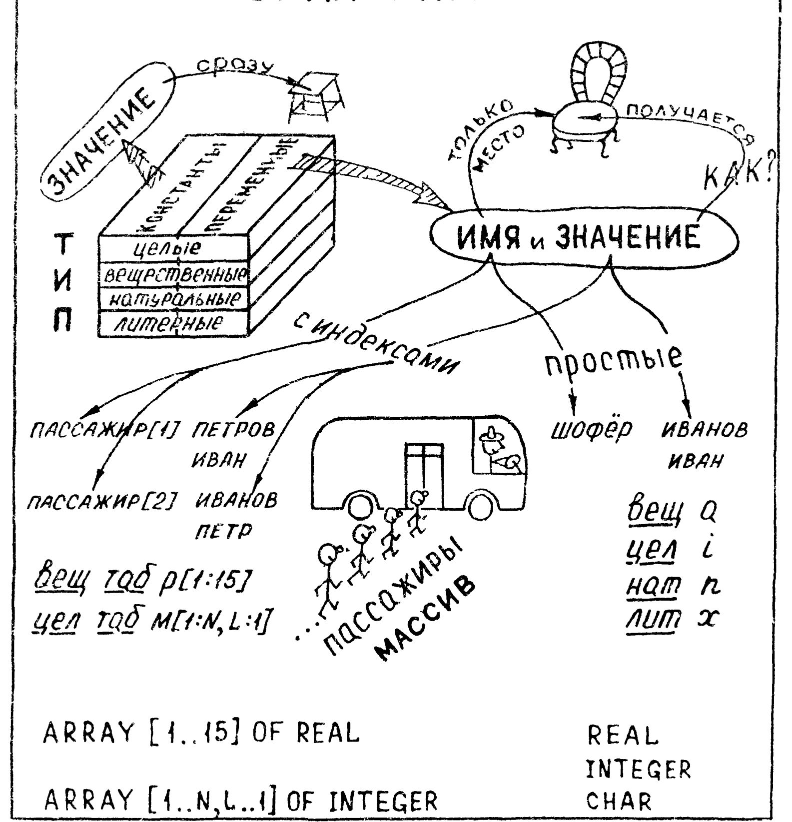

УПРАВЛЕНИЕ В АЛГОРИТМЕ
2
„Что вы привыкли делать дома
Единым махом, наугад,
Как люди пьют или едят,
Вам расчленят на три приема."
Иоганн В. Гёте
ВЕТВЛЕНИЕ
если IF
то THEN
иначе ELSE
СЛЕДО ВАНИЕ
ПОВТОРЕНИЕ
пока (для) WHILE (FOR)..DO
ТЕЛО BEGIN
цикла ТЕЛО Ц.
конец цикла END
ДА
НЕТ
УСЛОВИЕ
СЕРИЯ 1
СЕРИЯ 2
КОМАНДА 1
КОМАНДА 2
серия
ОПЕРАТОР
деньги есть
НЕТ
ДА
КУПИЛ
ТРАНЖИРА
ОБЖОРА
СЪЕЛ
понравилось
ДА
НЕТ
!
ПАРАМЕТР
ИНИЦИАЛИЗАЦИЯ

3

*„Здесь у меня столовая*
*Вся мебель в ней дубовая.*
*Вот это стул — на нем сидят.*
*Вот это стол — за ним едят.“*

*С.Я. Маршак*

# ВЕЛИЧИНЫ



5. Составьте фрагмент алгоритма, в котором переменной X присваивается значение 5, если значение другой переменной A находится в интервале [0, 8]; переменной X присваивается значение 10, если значение переменной A находится в интервале [10, 17]; переменной X присваивается значение 15, если значение переменной A не попало ни в один из указанных интервалов.

6. Сколько раз выполняется действие присваивания в теле внутреннего цикла?

```
X := 0
для I от 1 до 5
  для J от 2 до 13
    X := X + 1
  конец цикла по J
конец цикла по I.
```

```
X := 10
пока X < = 100
  для Z от 5 до 7
    Y := Y + 1
  конец цикла по Z
  X := X * X
конец цикла пока.
```

```
X := 0
для I от 1 до 5
  для J от I до 10
    X := X + I + J
  конец цикла по J
конец цикла по I.
```

## ПОЯСНЕНИЯ К ЛОС № 3

ЛОС № 3 посвящен классификации данных и описанию их характеристик. Для проведения классификации используется опорный сигнал в виде кубика. Передняя грань кубика делит данные на 4 класса – вещественные, целые, натуральные и литерные. Служебные слова, описывающие соответствующий тип, приведены в правом нижнем углу ЛОС.

Почему в ЭВМ применяется подобное деление данных по ТИПУ?

1. Данные разного типа имеют разное внутреннее представление в ЭВМ. Например, данное литерного типа занимает в памяти столько байтов, сколько содержит символов, так как один символ размещается в одном байте. Данное типа вещ почти всегда занимает 4 байта (32 двоичных разряда), а данное целого типа (цел) – 2 байта (16 двоичных разрядов).

2. Для данных разного типа цел очень часто вводятся дополнительные арифметические операции целочисленного деления и вычисления остатка от деления.

На верхней грани кубика – другая линия классификации. В процессе работы алгоритма данные могут оставаться ПОСТОЯННЫМИ. Это – КОНСТАНТЫ. Данные могут и изменять свое значение – это ПЕРЕМЕННЫЕ. Константа ОДНОЗНАЧНО, РАЗ и НАВСЕГДА задается своим значением. 3.14 – это только 3.14 и более ничего. Отводя ей место в памяти, ЭВМ сразу на это место помещает константу и запоминает (для себя) адрес этого места. Потом, в процессе работы алгоритма, она будет находить константу по ее адресу, который так и остается неизвестным программисту.

Переменная может принимать разные значения в процессе работы алгоритма, поэтому для нее и вводится обозначение – ИМЯ. Когда ЭВМ встречает в алгоритме произвольное имя, она отводит ему ТОЛЬКО место в памяти (опорный сигнал – СТУЛ). Значение же (то, что будет находиться на стуле) получается в различных командах алгоритма. Подчеркнем, что место отводится при первой встрече имени, т.е. один раз, значение же может меняться много раз. На ЛОС стоит вопрос КАК? Как можно менять значение одной и той же переменной? Существуют ДВЕ возможности – команда ВВОДА и команда ПРИСВАИВАНИЯ. В первом случае значение переменной берется с внешнего носителя. Во втором случае оно может вычисляться по достаточно сложному выражению внутри центрального процессора ЭВМ.

Опорный сигнал в виде кубика позволяет сделать вывод о возможности использования в алгоритмах как констант целого, вещественного, натурального и литерного типов, так и соответствующих переменных.

В нижней части ЛОС представлена еще одна линия классификации переменных величин – по СПОСОБУ ОРГАНИЗАЦИИ. Это ПРОСТЫЕ переменные и переменные с ИНДЕКСАМИ. ПРОСТЫЕ переменные в программировании называют иногда СКАЛЯРНЫМИ (т.е. единичными). Переменные с ИНДЕКСАМИ – это элементы совокупностей однотипных величин, называемых в программировании МАССИВАМИ.

Для пояснения сути этой класификации переменных величин на ЛОС использован опорный сигнал (образный элемент) в виде автобуса. ШОФЕР – это имя ПРОСТОЙ скалярной величины. В автобусе шофер – один. Но на одном автобусе могут работать разные ШОФЕРЫ. Следовательно, это имя может иметь разные значения. Для данного примера такими значениями будут – фамилия, имя и отчество каждого шофера (см. на ЛОС), т.е. значения литерного типа.

Автобус перевозит пассажиров. Они образуют МАССИВ переменных. Общее ИМЯ этого массива – ПАССАЖИР. В автобусе каждый пассажир полностью определяется номером места, которое он занимает (прочие качества пассажира здесь не имеют значения). Значит, к каждому из них можно обратиться следующим образом:

ПАССАЖИР [1],<br>ПАССАЖИР [2] и т.д.

Это и есть ИМЯ переменной с индексами.

Достаточно указать номер места (индекс), как нужный пассажир отзовется, и можно узнать его ЗНАЧЕНИЕ – фамилию, имя и отчество, т.е., как и в случае с шофером, значение литерного типа. Данная аналогия позволяет зафиксировать внимание на сущности понятия ИМЯ ПЕРЕМЕННОЙ, так как в самой аналогии слово ИМЯ используется дважды – как понятие программирования и как значение литерного типа. Различие этих двух моментов и говорит о понимании сути дела.

Таким образом, МАССИВЫ (ТАБЛИЦЫ) – это совокупности ОДНОТИПНЫХ переменных, в которых каждый элемент однозначно определяется своим местом в ряду элементов, образующих эту совокупность.

Слева внизу на ЛОС приведены примеры описаний.

## ВОПРОСЫ К ЛОС № 3

1. Какие характеристики величин вы можете назвать?

2. Чем характеризуется постоянная величина ?

3. Чем характеризуется переменная величина ?

4. Какие типы величин определены в алгоритмическом языке ?

5. Какие величины называются табличными (массивами) ?

6. Что значит переменная с индексами ?

7. Где описываются табличные величины (массивы) ?

8. Где указывается тип величин в записи алгоритма ?

9. Зачем в алгоритме дается описание величин ?

10. Опишите двумерный целый массив, состоящий из 3-х строк и 6 столбцов.

11. Как обратиться к элементу, стоящему на пересечении 2-й строки и 5-го столбца ?

12. Пусть в один элемент массива может быть записан один символ. Какого размера нужно описать символьный массив, чтобы в него можно было занести фразу: "Программирование зеркало разума" ?

## ПОЯСНЕНИЯ К ЛОС № 4

Выбранная структура ЛОС удобна для демонстрации возможностей представления данных в памяти. По желанию преподавателя можно выбрать те типы данных (игрушки на елочке), с которыми затем предполагается работать.

Вся совокупность данных делится на 2 части в зависимости от места хранения значений – в оперативной памяти или на внешних носителях в виде ФАЙЛОВ. Рассказ по ЛОС удобно вести, поднимаясь по елочке от простых типов к сложным. О файлах можно и не говорить, но при этом теряется полнота представления информации в памяти ЭВМ.

Данные, значения которых находятся в оперативной памяти, в свою очередь,тоже делятся на 2 группы – простые и сложные (структрированные) . Рекомендуется подробно разобрать простые типы. "Ключик" к ним поясняет, о чем конкретно сообщается для каждого типа: "тип" – служебное слово для описания: "const" – запись константы, "внутр" – способ внутреннего представления данных в памяти, "бит" – размер выделяемой для данного типа памяти (в битах). О сложных типах приводится только самая общая информация – описание переменной соответствующего типа и пример возможного значения. В частности, ЗАПИСЬ здесь – сведения о городе, а именно название, год основания и количество жителей. Для более детальной проработки сложных типов можно использовать ЛОС № 5.

TYRE – раздел описания типов данных в программе на Паскале.

## ВОПРОСЫ К ЛОС № 4

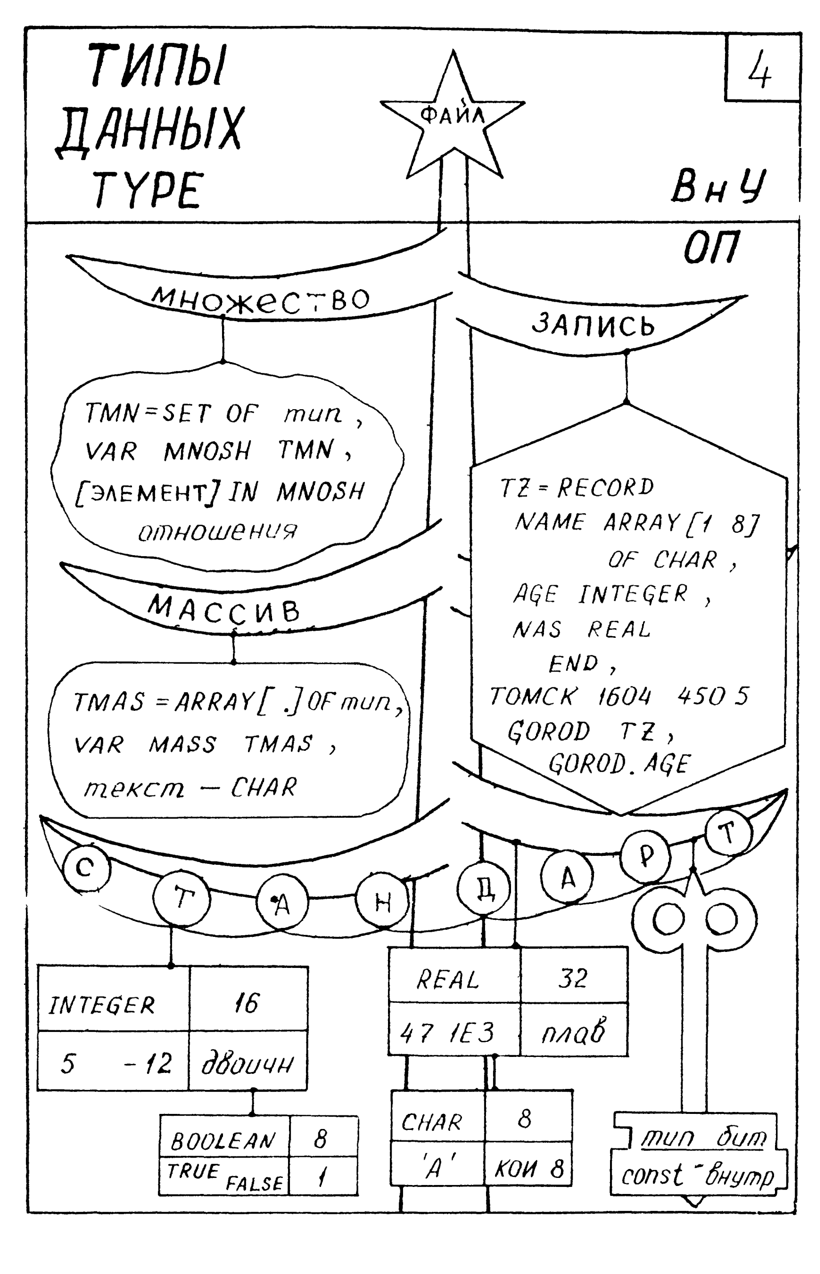
1. Что определяет тип данного ?

2. Сформулировать задачу, в которой исходная совокупность данных может быть представлена МАССИВОМ.

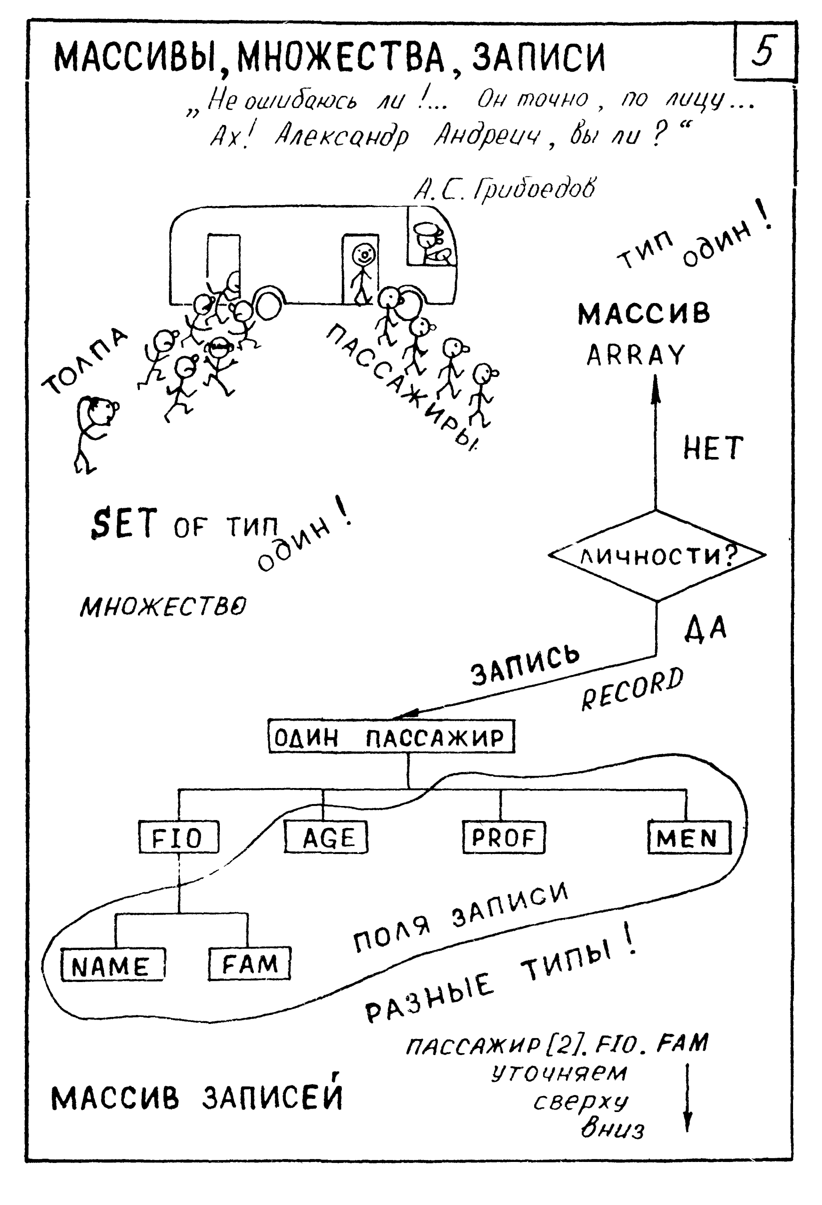
3. Сформулировать задачу, в которой исходная совокупность данных может быть представлена в виде ЗАПИСИ.

4. Сколько места в памяти выделяется для хранения 10 элементов а) целочисленного; б) вещественного; в) символьного МАССИВОВ?

5. Назовите известные вам внешние запоминающие устройства и носители информации.



ТИПЫ
ДАННЫХ
TYPE
4
ФАЙЛ
ВнУ
ОП
МНОЖЕСТВО
ЗАПИСЬ
TMN=SET OF тип ,
VAR MNOSH TMN ,
[ЭЛЕМЕНТ] IN MNOSH
отношения
TZ = RECORD
NAME ARRAY [1 8]
OF CHAR ,
AGE INTEGER ,
NAS REAL
END ,
TOMCK 1604 450 5
GOROD TZ ,
GOROD. AGE
МАССИВ
TMAS = ARRAY [ .] OF тип,
VAR MASS TMAS ,
текст – CHAR
С Т А Н Д А Р Т
INTEGER
16
5 -12
двоичн
REAL
32
47 1E3
плав
BOOLEAN
8
TRUE FALSE
1
CHAR
8
'A'
КОИ 8
тип бит
const – внутр


МАССИВЫ, МНОЖЕСТВА, ЗАПИСИ
5
„Не ошибаюсь ли !... Он точно, по лицу...
Ах! Александр Андреич, вы ли ?“
А.С. Грибоедов
ТОЛПА
ПАССАЖИРЫ
тип один !
МАССИВ
ARRAY
НЕТ
SET OF тип один !
МНОЖЕСТВО
ЛИЧНОСТИ?
ДА
ЗАПИСЬ
RECORD
ОДИН ПАССАЖИР
FIO
AGE
PROF
MEN
NAME
FAM
ПОЛЯ ЗАПИСИ
РАЗНЫЕ ТИПЫ !
ПАССАЖИР [2]. FIO. FAM
уточняем
сверху
вниз
МАССИВ ЗАПИСЕЙ

6. Когда возникает необходимость хранить данные на внешнем носителе ?

## ПОЯСНЕНИЯ К ЛОС № 5

ЛОС № 5 предназначен для совместного рассмотрения и обобщения знаний по сложным структурам данных – МАССИВАМ, ЗАПИСЯМ и МНОЖЕСТВАМ. За основу взята ассоциация с ЛОС № 3 – пассажиры. Их упорядоченная последовательность, когда объединение элементов в общую структуру идет только по одному какому-то признаку, одинаковому для всех (типу), является МАССИВОМ. Неупорядоченная совокупность элементов одного типа, рассматриваемая как куча или толпа, является МНОЖЕСТВОМ. И, наконец, объединение в одно целое элементов разного типа представляет собой ЗАПИСЬ. На ЛОС № 3 для представления типа ЗАПИСЬ используется факт рассмотрения пассажиров в качестве личностей. О каждом пассажире в поля ЗАПИСИ фиксируются: имя и фамилия (NAME, FAM), возраст (AGE), профессия (PROF) и признак (MEN), принимающий значение ИСТИНА, если пассажир – мужчина, и ЛОЖЬ, если – женщина. Ниже приведено описание подобной записи на языке Паскаль и указаны возможные значения полей записи:

```
TYPE
 L=RECORD
     FIO : RECORD
           NAME : ARRAY [1..8] OF CHAR ;       ВЛАДИМИР
           FAM : ARRAY [1..12] OF CHAR ;       РУКАВИШНИКОВ
         END ;
     AGE: INTEGER ;                            28
     PROF : ARRAY [1..12] OF CHAR ;            ПРОГРАММИСТ
     MEN : BOOLEAN                             ИСТИНА
    END.
```

Переменная типа ЗАПИСЬ объявляется как обычно: VAR A : L.

Обращение к любому элементу ЗАПИСИ выполняется с помощью конструкции, которая носит название уточняющего имени. Для приведенного примера

элемент A. FIO. NAME [6] имеет значение символа "M",
элемент A. AGE – значение 28,
элемент A. PROF [12] – значение символа " ",
элемент A. MEN – значение ИСТИНА.

## ВОПРОСЫ К ЛОС № 5

1. Занести в память ЭВМ в виде ЗАПИСИ следующие сведения о вашей любимой книге: фамилия автора, название книги, год издания, количество страниц.

2. Занести в память ЭВМ в виде МАССИВА ЗАПИСЕЙ вышеперечисленные сведения обо всех книгах вашей домашней библиотеки.

3. Вы поменялись с другом книгами одного автора. Изменить в памяти ЭВМ сведения о вашей домашней библиотеке.

4. В ваш класс приходит новый ученик. Проверить, есть ли в классе

ученики с тем же именем, что у новичка, при условии:

а) все имена представлены в виде МАССИВА имен;

б) все имена представлены в виде МНОЖЕСТВА имен;

в) все имена содержатся в МАССИВЕ ЗАПИСЕЙ об учениках.

5. Сколько места в памяти ЭВМ занимает информация об учениках класса, представленная в виде значения переменной INF ?

```
TYPE  UCH = RECORD
                FAM : ARRAY [1 .. 12] OF CHAR ;
                ROST : INTEGER ;
                VES : REAL
              END ;
VAR  INF : ARRAY [1 .. 42 ] OF UCH.
```

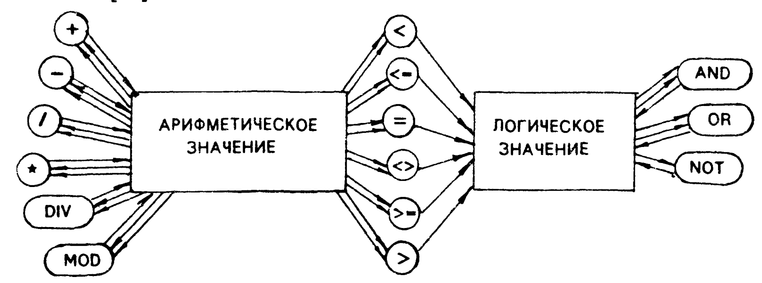
## ПОЯСНЕНИЯ К ЛОС № 6

На ЛОС № 6 приведены сведения об операциях над данными разных типов. Все перечисленные операции реализованы в языке Паскаль, некоторые из них можно использовать при записи алгоритмов на щкольном алгоритмическом языке. В процессе работы с этим ЛОС рекомендуется обратить внимание на следующие моменты.

1. Операции отношения выполняются над теми типами данных, значения которых можно упорядочить. Сравнение символа (например, 'A' > 'B') производится посредством сравнения их внутренних кодов (ASCII, EBCDIC,КОИ-8).

2. В Паскале для деления целочисленных данных разрешены только операции DIV и MOD.

3. Разобраться с логическими операциями поможет диаграмма, взятая из [11]:




4. Достаточно подробно логические операции разбираются в задаче "Расписание уроков".

5. Операции над множествами рассматриваются в задаче "Спортлото". Именно в тесной связи со способами ее решения следует изучать этот материал.

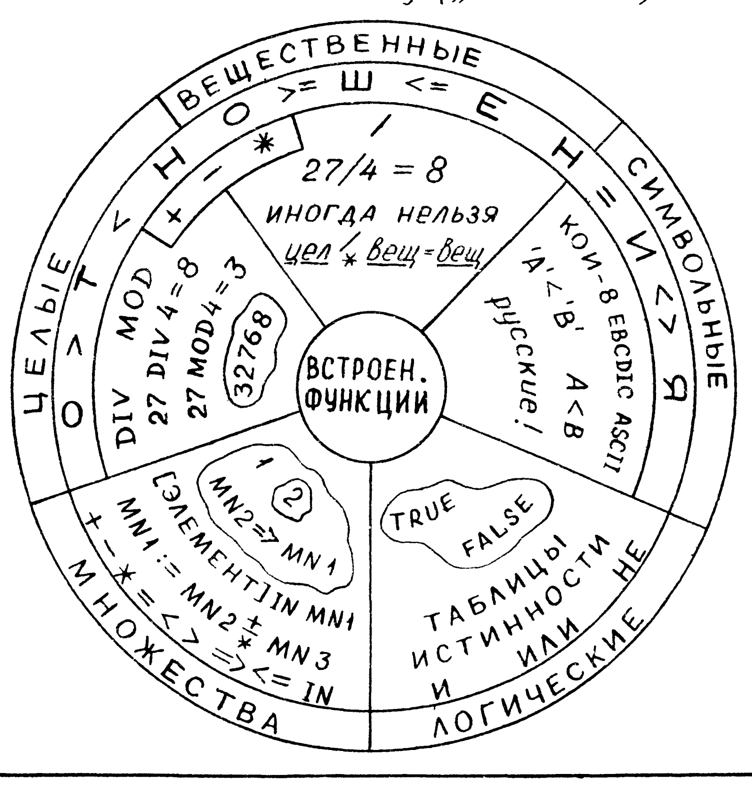
6. В сложных выражениях могут встретиться разные операции. Для сохранения правильной последовательности вычислений нужно расставлять круглые скобки и учитывать приоритет выполнения операций. В языке Паскаль операции упорядочены (по убыванию приоритета):

# ОПЕРАЦИИ

6

*„Но много ли среди нас таких, которые хотя бы знали толком, что означают все эти названия?“*

*Б. Шоу („Пигмалион“)*



NOT,[* , / , DIV, MOD, AND], [+ , – , OR], [< , <= , > , >= , = , <> ]. В квадратных скобах объединены операции с одним приоритетом, они выполняются слева направо.

7. В каждом языке программирования имеется определенный набор стандартных, так называемых встроенных функций для разных типов данных. Нужно ознакомиться с ними, чтобы затем использовать в программах.

## ВОПРОСЫ К ЛОС № 6

1. Указать последовательности вычисления значений выражений:

A + B/C – 4.5 ; A + B DIV C – 4.5 ;
D – F + A MOD B ; (A + B) – (D DIV C) MOD A.

2. Переписать формулы в виде выражений алгоритмического языка:

$$\frac{1+a}{1+b} - \frac{1+c}{1+d}; \qquad \frac{1}{a+b} + \frac{1}{a} \cdot \frac{2-c}{2-d};$$

$$\frac{1+a}{1+\frac{a}{1+a}}; \qquad \frac{a \cdot b \cdot c}{d \cdot f - 1}.$$

3. Пояснить выполнение операций DIV и MOD.
4. Что больше а) A или P; б) 'A' или 'P'?
5. Как ЭВМ сравнивает между собой символы?
6. Какой тип должен быть у переменных в приведенном ниже выражении, чтобы оно было корректным?

(A MOD C > 10) AND P.

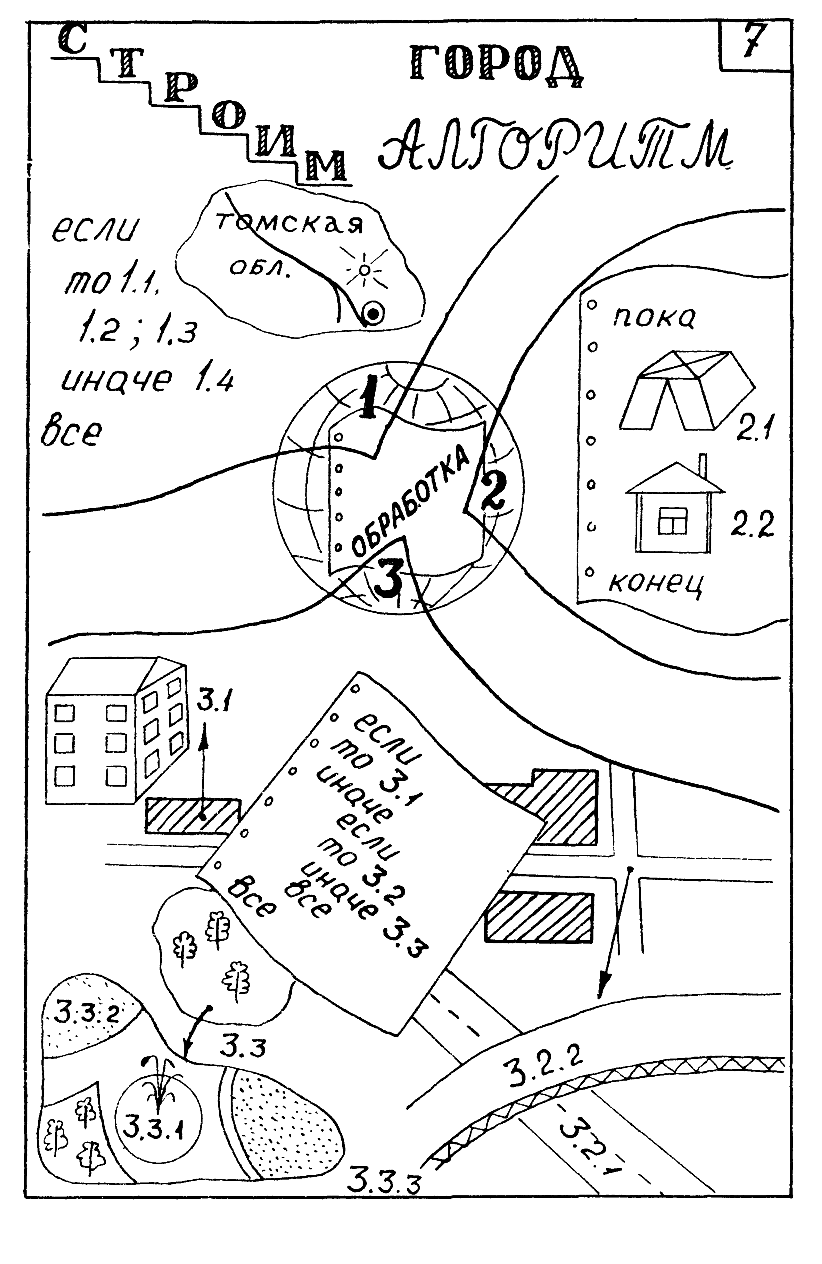
## ПОЯСНЕНИЯ К ЛОС № 7

*Дедукция — умозаключение от общего к частному, от общих суждений к частным и другим общим выводам.*

*Искусство делать выводы и анализировать, как и все другие искусства, постигается долгим и прилежным трудом, но жизнь слишком коротка, и поэтому ни один смертный не может достичь полного совершенства в этой области. Прежде чем обратиться к моральным и интеллектуальным сторонам дела, которые представляют собой наибольшие трудности, пусть исследователь начнет с более простых задач...*

*...Правила дедукции... просто бесценны для моей практической работы.*

***Артур Конан Дойль,***
*"Этюд в багровых тонах"*


7
СТРОИМ ГОРОД АЛГОРИТМ
если
то 1.1,
1.2 ; 1.3
иначе 1.4
все
томская
обл.
1
ОБРАБОТКА
2
3
пока
2.1
2.2
конец
3.1
если
то 3.1
иначе
если
то 3.2
иначе 3.3
все
все
3.3.2
3.3
3.3.1
3.3.3
3.2.2
3.2.1

Можно провести аналогию между разработкой алгоритма решения задачи на ЭВМ и планированием этапов строительства нового города. Обратимся к ЛОС № 7.

Пусть где-то на Земле нужно построить новый город. Исходными данными могут быть желаемые параметры (географические, экономические, социальные и т.д.) этого города: обработкой – все, что связано со строительством; результаты – реально построенный город.

Разделим процесс строительства на три этапа: 1 – изыскания; 2 – подготовка, 3 – стройка. Начиная с этого разделения, этапы можно разрабатывать, доводить до частных действий отдельно один от другого. Пользуясь алгоритмической нотацией, запишем, например:

```
1 --> ЕСЛИ место стройки Томская область
      ТО
         найти площадку для строительства на
         берегу Оби                                  ---->1.1
         произвести испытания грунта                 ---->1.2
         разработать проект стройки                  ---->1.3
      ИНАЧЕ
         перенести строительство на другой срок      ---->1.4
      ВСЕ.
```

Здесь парами цифр обозначены действия, являющиеся детализацией этапа 1. Каждое из них требует дальнейшего более частного рассмотрения, которое снова можно провести независимо от других действий.

Перейдем к разработке этапа 2 – подготовка. Конечно, в действительности он состоит из очень многих действий, мы же ограничимся одним:

```
2 -->ПОКА не все строители имеют жилье
        поставить палатку               ---->2.1
        построить времянку              ---->2.2
     КОНЕЦ цикла ПОКА.
```

Этот этап можно было бы описать фразой: "Построить временное жилье для всех строителей", но затем ее все равно пришлось бы детализировать в виде цикла, поскольку весь процесс разработки алгоритма направлен на получение в конечном счете последовательности алгоритмических конструкций.

Этап 3 – стройка. Детализируя это действие, мы будем пользоваться результатами выполнения действий 1 и 2, так как они полностью завершатся к моменту начала выполнения действия 3. Предположим, что этим действием будет строительство микрорайона. В какой последовательности возводить отдельные объекты? Если принять за правило, что сначала строятся жилые дома, затем оборудуются магистрали, в последнюю очередь – скверы, то можно детализировать действие 3 следующим образом:

```
3 -->ЕСЛИ жилья не хватает
     ТО построить многоквартирный дом          ---->3.1
     ИНАЧЕ
```

```
    ЕСЛИ движение интенсивное
    ТО построить дорожную развязку        ---->3.2
    ИНАЧЕ разбить сквер                   ---->3.3
    ВСЕ
  ВСЕ.
```

Каждое из новых конкретных действий можно, в свою очередь, детализировать. Например, для строительства дома:

```
3.1 --> выбрать проект                          ---->3.1.1
        подготовить стройплощадку               ---->3.1.2
        выполнить работы нулевого цикла         ---->3.1.3
        сложить коробку дома                    ---->3.1.4
        провести отделочные работы              ---->3.1.5
```

Для организации развязки на дороге:

```
3.2 --> проложить  магистраль          ---->3.2.1
        построить эстакаду             ---->3.2.2
```

Для озеленения города:

```
3.3 --> установить фонтан              ---->3.3.1
        разбить цветники               ---->3.3.2
        проложить дорожки              ---->3.3.3
```

Объединив все шаги детализации в единое целое, получим общую структуру алгоритма строительства города:

```
       ВВОД
1 -->  ЕСЛИ
       ТО
              1.1
              1.2
              1.3
       ИНАЧЕ  1.4
       ВСЕ
2 -->  ПОКА
              2.1
              2.2
       КОНЕЦ ПОКА
3 -->  ЕСЛИ
       ТО
              3.1.1
              3.1.2
              3.1.3
              3.1.4
              3.1.5
       ИНАЧЕ    ЕСЛИ
                ТО
                      3.2.1
                      3.2.2
```

Чем больше цифр в обозначении действия, тем глубже уровень детализации, на котором это действие рассматривается.

ИНАЧЕ
3.3.1
3.3.2
3.3.3
ВСЕ
ВСЕ
ВЫВОД.

Пользуясь теми же правилами разработки последовательности действий от общего к частному, можно конструировать алгоритмы решения задач на ЭВМ. Абстрагирование от частностей, мелких подробностей на начальных этапах разработки алгоритма позволяет определить его общее строение и оформить его таким образом, чтобы в целом следование действий друг за другом образовывало логически законченную структуру.

Использование метода пошаговой детализации позволяет проследить развитие алгоритма вглубь, подчеркнуть, что алгоритм – это последовательность конструкций, каждая из которых может быть достаточно богатой своими "внутренними" действиями. Примерами могут служить организация вложенных циклов или цепочек ЕСЛИ–ТО–ИНАЧЕ.

## ПОЯСНЕНИЯ К ЛОС № 8

Самым житейским примером, который может быть ассоциативно использован для пояснения способа записи алгоритма, является любой кулинарный рецепт: КАК приготовить вполне определенное блюдо (ЧТО) из вполне определенных продуктов (ИЗ ЧЕГО). Аналогия, можно сказать полная. ИМЯ алгоритма – это НАЗВАНИЕ производимого продукта. Если алгоритм пишется для ЭВМ (т.е. составляется ПРОГРАММА на определенном алгоритмическом языке), то его в наше время так и называют – программный продукт. Указание типов имен, являющихся аргументами (входами) и результатами (выходами) алгоритма – это задание конкретных характеристик величин, для обработки которых предназначен алгоритм. И, наконец, последовательность команд (операторов) от начала до конца – это последовательность действий, необходимых для получения результата.

Вторая часть ЛОС № 8 посвящена вопросам оптимального использования труда программистов. Дело в том, что хороший алгоритм, составленный однажды, не имеет тенденции со временем портиться, и потому им можно пользоваться как угодно долго. Можно его просто каждый раз при надобности переписывать, но это не всегда можно сделать быстро, и поэтому в программировании пошли по другому пути.

Из хороших алгоритмов составляют БИБЛИОТЕКИ АЛГОРИТМОВ и определяют общие правила пользования такими библиотеками. Все алгоритмы, занесенные в библиотеки, называются ВСПОМОГАТЕЛЬНЫМИ. В алгоритмических языках они называются ПРОЦЕДУРАМИ и ФУНКЦИЯМИ. Чтобы воспользоваться вспомогательным алгоритмом при написании другого алгоритма (на ЛОС последний помещен в большую букву А и назван – ОСНОВНОЙ АЛГОРИТМ; его имя для примера тоже взято А), необходимо в нужном месте написать команду (оператор) ВЫЗОВА

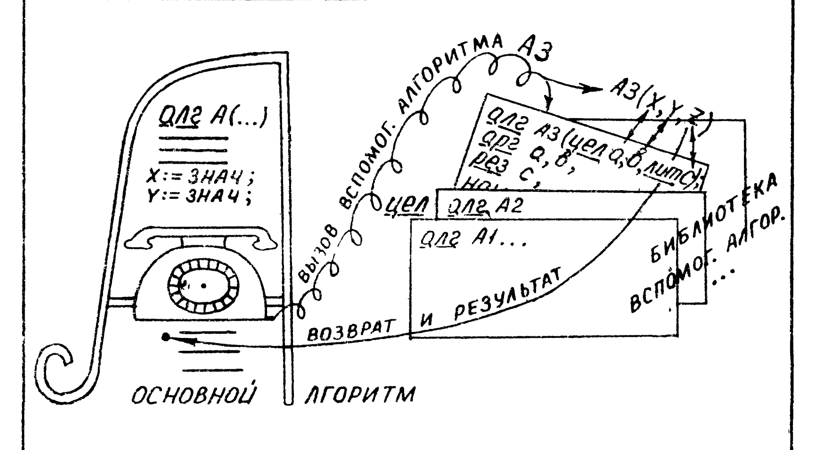
8

„Изучая поваренную книгу, программист может узнать много интересного и полезного."

Д. Кнут

# ЕЩЁ РАЗ ОБ АЛГОРИТМАХ

| РЕЦЕПТ | | ПРОГРАММА |
|---|---|---|
| ТОРТ название | ЧТО? | алг имя алгоритма |
| сахар<br>масло<br>⋮ | ИЗ ЧЕГО? | (тип имя1, имя2, тип имя3.)<br>арг имя1, имя2, ...<br>рез имя3, ... |
| растереть<br>добавить | КАК? | нач<br>оператор 1;<br>оператор 2, ...<br>кон |




PROGRAM

PROCEDURE

FUNCTION

вспомогательного алгоритма. Эта команда имеет следующий формат:

Имя вспом. алг. (список конкретных имен-аргументов, для которых должен проработать вспомогательный алгоритм).

Это показано на ЛОС № 8. К моменту вызова (опорный сигнал в виде телефона) вспомогательного алгоритма с именем А3 в основном алгоритме должны быть обязательно описаны и определены значения для переменных, которые при вызове будут переданы как аргументы. Должна быть описана также и величина, которая будет передана во вспомогательный алгоритм для получения результата его работы.

Работа со вспомогательными алгоритмами требует от программиста внимательности и аккуратности. Сделаем несколько замечаний, важных в данном случае.

Первое очевидное замечание –
все величины, передаваемые из основного алгоритма во вспомогательный, должны соответственно совпадать по типу. Сопоставляются они по порядку следования, что отражено на ЛОС стрелочками.

Второе очевидное замечание –
алгоритм, вспомогательный в одном случае, может быть, в свою очередь, основным по отношению к другому алгоритму, если во время своей работы он обращается к нему. Этот другой алгоритм будет по отношению к первому вспомогательным.

Третье очевидное замечание –
на ЛОС № 8 стрелочкой с надписью ВОЗВРАТ и РЕЗУЛЬТАТ подчеркнут тот факт, что возвращение результата работы вспомогательного алгоритма происходит всегда в вызывающий, т.е. основной для него, алгоритм к команде, следующей за командой вызова.

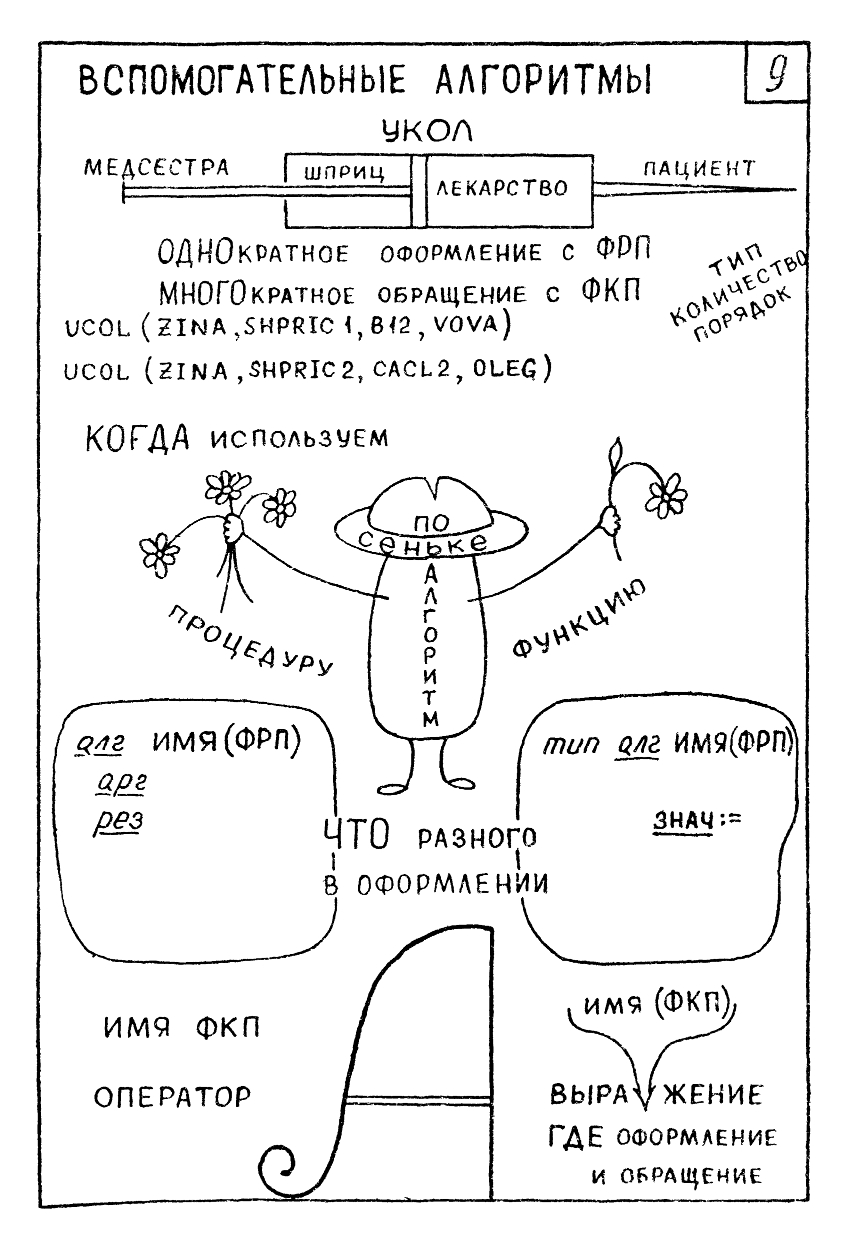
## ПОЯСНЕНИЯ К ЛОС № 9

На ЛОС № 9 представлены сведения о вспомогательных алгоритмах в школьном алгоритмическом языке и языке Паскаль. Опорный сигнал в виде шприца подчеркивает, что вспомогательный алгоритм предназначен для многоразового выполнения одних и тех же действий (УКОЛ) над разными совокупностями входных данных и, соответственно, с разными значениями получаемых результатов.

При ОФОРМЛЕНИИ вспомогательных алгоритмов в роли исходных данных и результатов выступают ФОРМАЛЬНЫЕ параметры: МЕДСЕСТРА, ШПРИЦ, ЛЕКАРСТВО, ПАЦИЕНТ – переменные, не имеющие значений в памяти ЭВМ. На время выполнения вспомогательного алгоритма они заменяются ФАКТИЧЕСКИМИ параметрами (аргументами): ZINA, SHPRIC1, SHPRIC2, B12, CACL2, VOVA, OLEG (их значениями или адресами). Таким образом, осуществляется информационная связь между отдельными алгоритмами. Работая со вспомогательными алгоритмами, нужно помнить об определенных правилах: формальные и фактические параметры должны совпадать по количеству, порядку следования и типу.

В алгоритмических языках используется 2 вида вспомогательных алгоритмов:

ФУНКЦИИ – в ситуации, когда результатом работы алгоритма является одно значение (цветок);


ВСПОМОГАТЕЛЬНЫЕ АЛГОРИТМЫ
9
УКОЛ
МЕДСЕСТРА
ШПРИЦ
ЛЕКАРСТВО
ПАЦИЕНТ
ОДНОкратное оформление с ФРП
МНОГОкратное обращение с ФКП
ТИП
КОЛИЧЕСТВО
ПОРЯДОК
UCOL (ZINA, SHPRIC 1, B12, VOVA)
UCOL (ZINA, SHPRIC 2, CACL 2, OLEG)
КОГДА используем
ПО
сеньке
АЛГОРИТМ
ПРОЦЕДУРУ
ФУНКЦИЮ
алг имя (ФРП)
арг
рез
тип алг имя(ФРП)
знач :=
ЧТО разного
в оформлении
имя ФКП
ОПЕРАТОР
имя (ФКП)
ВЫРАЖЕНИЕ
ГДЕ оформление
и обращение

ПРОЦЕДУРЫ– в ситуации, когда результатом работы алгоритма является любое число значений (букет).

Указанные различия влекут за собой и различие в оформлении и использовании этих алгоритмов, что представлено на ЛОС.

Для того, чтобы вспомогательный алгоритм выполнился, нужно осуществить его ВЫЗОВ, указав при этом фактические параметры, над которыми будут выполнены действия. На ЛОС пояснено, как это делается для разных видов вспомогательных алгоритмов – процедур и функций. И, наконец, на ЛОС приведена общая структура программы, содержащей оформление (описание) вспомогательных алгоритмов и их вызов.

## ВОПРОСЫ К ЛОС № 9

1. Зачем необходимо оформлять алгоритм в виде процедур или функций?

2. Что такое ФОРМАЛЬНЫЕ параметры?

3. Что такое ФАКТИЧЕСКИЕ параметры (аргументы)?

4. Укажите, в чем разница обращения к процедурам и функциям.

5. Какими должны быть соотношения между формальными и фактическими параметрами?

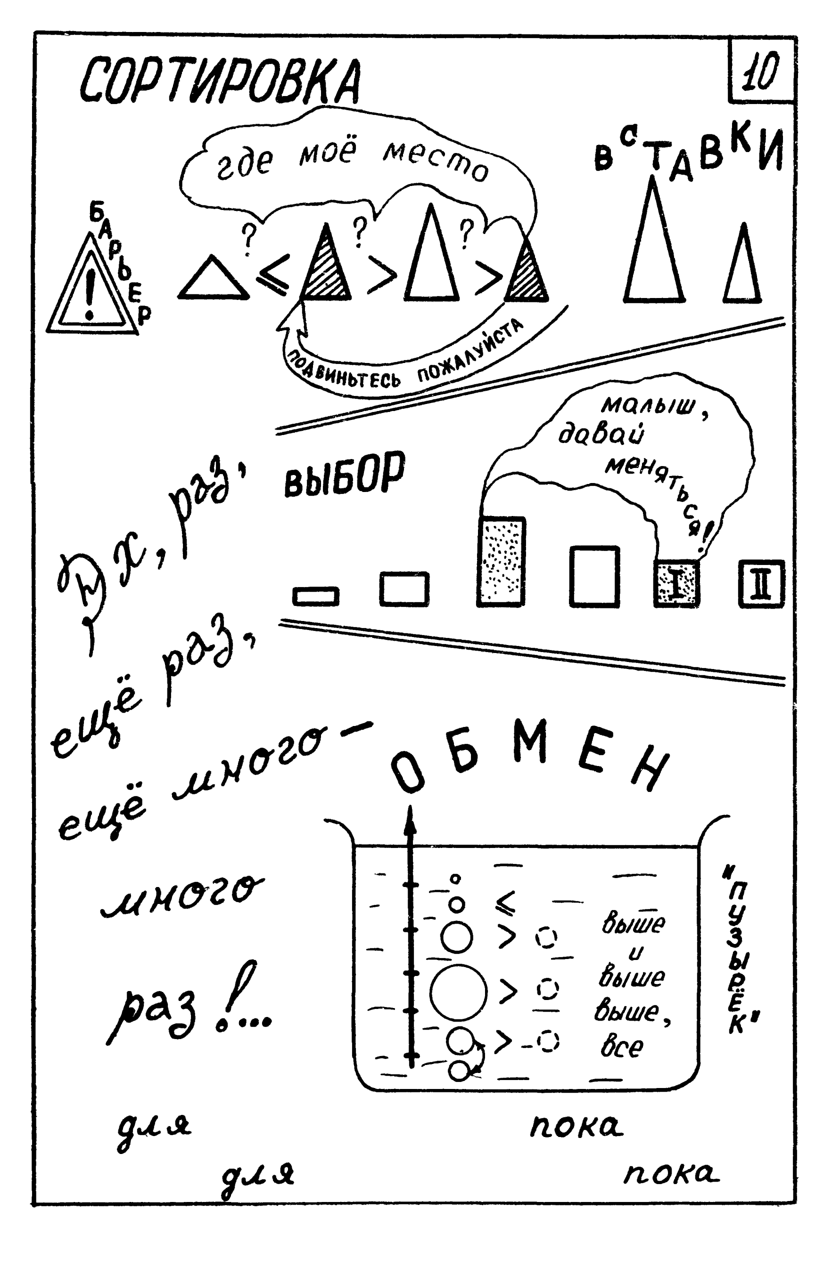
6. Оформите в виде процедуры и в виде функции алгоритм определения наибольшего общего делителя двух чисел.

7. Напишите программу, в которой наибольший общий делитель нескольких пар целых чисел вычисляется как путем обращения к процедуре, так и путем обращения к функции.

## ПОЯСНЕНИЯ К ЛОС № 10

С помощью ЛОС № 10 можно разобрать идею простейших методов сортировки последовательностей значений. Данные, подлежащие упорядочению, должны храниться в памяти в виде массивов. Если это массив записей, то упорядочение ведется по некоторому ключевому полю. На ЛОС размеры геометрических фигур соответствуют величинам элементов массива. Договоримся, что упорядочение ведется по возрастанию, от меньшего к большему.

Метод вставок. Предположим, что несколько первых элементов массива уже упорядочены (первые 3 треугольника). Найдем место для 4-го элемента в этой части последовательности. Очевидно, он должен встать после того элемента, который меньше или равен ему. Получается, что часть последовательности от найденного места до 4-го элемента должна сдвинуться на 1 позицию вправо. Подобные сравнения и сдвиги нужно выполнять для всех элементов массива. Опорный сигнал в виде дорожного знака "Прочие опасности" напоминает о том, что при поиске места для некоторого элемента в упорядоченной части последовательности имеется возможность выхода за левую границу массива. Избежать этой опасности можно либо введя дополнительную проверку номера элемента ($I > 0$), либо удлинив массив за счет фиктивного нулевого элемента с номером 0, помещая в него заранее значение анализируемого элемента. Такой подход называется методом "барьера", он подробно рассматривается в задаче "Компьютер для Вильяма Леграна", но для другого способа сортировки.


СОРТИРОВКА
10
где моё место
вставки
БАРЬЕР
ПОДВИНЬТЕСЬ ПОЖАЛУЙСТА
малыш, давай меняться!
ВЫБОР
Эх, раз, ещё раз, ещё много-много раз!...
ОБМЕН
выше и выше выше, все
"ПУЗЫРЁК"
для
пока
для
пока

Метод выбора. Предполагается, что часть последовательности уже упорядочена. Какой элемент должен встать на следующее место? В оставшейся неупорядоченной последовательности отыскивается минимальный элемент, и он меняется местами с первым элементом неупорядоченной части. Такие действия выполняются над всеми элементами последовательности.

Метод обмена ("пузырька"). Пусть последовательность рассматривается с конца. Последний и предпоследний элементы сравниваются друг с другом и в случае необходимости (последний < предпоследнего) меняются местами. Затем сравниваются предпоследний и стоящий перед ним элементы, если нужно, они тоже меняются местами. И такое сравнение двух соседних элементов повторяется до начала последовательности. В результате самый маленький элемент переместится ("всплывет") на первое место, в дальнейших сравнениях он уже не будет участвовать.

Опорный сигнал "Эх, раз..." говорит о том, что для упорядочения последовательностей простейшими методами необходимо многократное сравнение их элементов друг с другом. Обычно цели можно достичь с помощью двух вложенных циклов, т.е. для последовательности из N элементов нужно выполнить порядка N ★ N сравнений.

## ЗАДАНИЯ К ЛОС №10

1. Разработать алгоритм сортировки вставками, используя метод пошаговой детализации (ЛОС №7). Учесть, что в этой сортировке можно выделить 2 независимых этапа – поиск места и сдвиг части последовательности.

2. Разработать алгоритм сортировки выбором в двух вариантах: а) отыскивая максимальный элемент; б) отыскивая минимальный элемент.

3. Разработать алгоритм сортировки "пузырьком" в двух вариантах: а) "пузырек" всплывает; б) "пузырек" тонет.

4. Модифицировать полученные алгоритмы для упорядочения по убыванию.

6. Составить программу, упорядочивающую список учеников вашего класса.

# ЛИТЕРАТУРА

*Ты жалуешься, что тебе там не хватает книг. Но ведь дело не в том, чтобы книг было много, а в том, чтоб они были хорошие: от чтенья с выбором мы получаем пользу, от разнообразного — только удовольствие. Кто хочет дойти до места, тот выбирает одну дорогу, а не бродит по многим, потому что это называется не идти, а блуждать.*

***Луций Анней Сенека,***
*"Нравственные письма к Луцилию"*


1. **Абрамов С.А., Гнездилова Г.Г., Капустина Е.Н., Селюн М.И.** Задачи по программированию. – М. : Наука, 1988. – 224 с.
2. **Александрович Н.Ф.** Занимательная грамматика. – Минск: Народная асвета, 1964. – 252 с.
3. **Алексеев В.Е., Ваулин А.С.** Задачи и упражнения по программированию. – М. : Высшая школа, 1989. – 112 с.
4. **Белошапка В., Лесневский А.** Основы информационного моделирования // Информатика и образование. – 1989. – № 3. С. 17–24.
5. **Боон К.** Паскаль для всех. – М. : Энергоатомиздат, 1988. – 189 с.
6. **Вильямс Р., Маклин К.** Компьютеры в школе. – М. : Прогресс, 1988. – 335 с.
7. **Вирт Н.** Алгоритмы + структуры данных = программы. – М. : Мир, 1985. – 406 с.
8. **Гарднер М.** Есть идея! – М. : Мир, 1982. – 304 с.
9. **Гарднер М.** Крестики-нолики. – М. : Мир, 1988. – 352 с.
10. **Гик Е.Я.** Занимательные математические игры. – М.: Знание, 1987. – 159 с.
11. **Григас Г.** Начала программирования. – М. : Просвещение, 1987. – 60 с.
12. **Грогоно П.** Программирование на языке Паскаль.–М.: Мир, 1982.– 254 с.
13. **Громов Г.Р., Шмелев А.Г., Гангнус П.А.** Компьютерные игры. – М.: Знание, 1988. – 95 с.
14. **Грэхем Р.** Практический курс языка Паскаль для микроЭВМ. – М.: Радио и связь, 1986. – 200 с.
15. **Дейкстра Э.** Дисциплина программирования. – М. : Мир, 1978. – С. 26.
16. **Жигарев А.Н., Макаров Н.В., Путинцева М.А.** Основы компьютерной грамоты. – Ленинград: Машиностроение, 1987. – 255 с.
17. **Журавлев А.П.** Языковые игры на компьютере. – М. : Просвещение, 1988. – 144 с.
18. **Закревский А.Д.** Алгоритмы синтеза дискретных автоматов. – М.: Наука, 1971. – 511 с.
19. **Касаткин В.Н.** Логическое программирование в занимательных задачах. – Киев: Техника, 1980. – 79 с.

20. **Касаткин В.Н., Владыкина Л.И.** Алгоритмы и игры. – Киев: Радяньска школа, 1984. – 80 с.
21. **Касаткин В.Н.** Необычные задачи математики. – Киев: Радянська школа, 1987. – 127 с.
22. **Клини С.** Математическая логика. – М.: Мир, 1973. – 478 с.
23. **Косневски Ч.** Занимательная математика и персональный компьютер. – М.: Мир, 1987. – 192 с.
24. **Костюк Ю.Л.** Вводный курс программирования. – Томск: Изд-во Том. ун-та, 1984. – 190 с.
25. **Липский В.** Комбинаторика для программистов. – М.: Мир, 1988.–213 с.
26. **Литцман В.** Веселое и занимательное о числах и фигурах. – М.: Физматгиз, 1963. – 264 с.
27. **Лэнгсман Й., Огенстайн М., Тененбаум А.** Структуры данных для персональных ЭВМ. – М.: Мир, 1989. – 568 с.
28. **Миллс Х., Хьюз Дж., Мичтом Дж.** Структурный подход к программированию. – М.: Мир, 1980. – С. 293.
29. **Мичи Д., Джонстон Р.** Компьютер – творец. М. : мир, 1987. – 254 с.
30. **Мочалов Л.П.** Головоломки. (Библиотечка "Квант", вып. 6). – М.: Наука, 1980. – 126 с.
31. **Нагибин Ф.Ф.** Математическая шкатулка. – М.: Учпедгиз, 1961. – 167 с.
32. **Наука** и жизнь. 1977. № 7. С. 122.
33. **Наука** и жизнь. 1985. № 1. С. 152.
34. **Наука** и жизнь. 1985. № 3. С. 127.
35. **Наука** и жизнь. 1987. № 6. С. 112.
36. **Перегудов М.А., Халамайзер А.Я.** Бок о бок с компьютером. – М.: Высшая школа, 1987. – 192 с.
37. **Персональный** компьютер в играх и задачах / Автор пред. П-26 И.М. Макаров. – М.: 1988. – 192 с.
38. **Петрович Н.Т.** Люди и биты. М.: Знание, 1986. – 190 с.
39. **Поддубная Т.Н.** Учение и обучение с опорными сигналами (на примере курса "Программирование на языке Паскаль"). – Томск: Изд-во Том. ун-та. 1986. – 195 с.
40. **Поддубная Т.Н.** Информатика по опорным сигналам // Информатика и образование. 1987. – № 3. – С. 23–38.
41. **Поддубная Т.Н, Фукс И.Л.** Построение алгоритмов // Информатика и образование. 1987. – № 5. – С. 64–75.
42. **Самарский А.А., Михайлов А.П.** Компьютеры и жизнь. – М.: Педагогика, 1987. – 128 с.
43. **Самохвалов Э.Н., Филиппович Ю.Н., Ревунков Г.И.** Задачи и упражнения по программированию. Ч. 1. – М.: Высшая школа, 1989. – 96 с.
44. **Светозарова Г.И., Мельников А.А., Козловский А.В.** Практикум по программированию на языке Бейсик. – М.: Наука, 1988. – 368 с.
45. **Ускова О.Ф., Горбенко О.Д.** Задачник-практикум по алгоритмическому языку. – Воронеж: Изд-во Воронеж. ун-та. 1987. – 142 с.
46. **Фукс И.Л.** Алгоритмы поиска и сортировки // Информатика и образование. 1988. – № 2. – С. 6 – 21.
47. **Шаталов В.Ф.** Точка опоры. – М.: Педагогика, 1987. – 160 с.
48. **Штейнгауз Г.** Сто задач. – М.: Наука, 1986. – 143 с.
49. **Daines D.** 26 BASIC programs for your micro. – Butterworths and Co. (Publishers) Ltd. 1983. Borough Green, Sevenoaks, Kent TN15 8PH. – 124 c.

## СОДЕРЖАНИЕ

Подписано в печать с РОМ 25.02.92 Формат 84x108/32 Бумага газетная
Гарнитура "Цюрих" Печать офсетная Усл. печ. л. 7,44
Усл. кр.-отт. 7,565 Уч.-изд. л. 9,68 Тираж 105000 экз. Изд. № 041
Заказ № 3581 Цена отпускная свободная.

Малое государственное редакционно-издательское предприятие "РАСКО". 634055, Томск, а/я 2211

Отпечатно в производственно-издательском комбинате "Офсет". 660049, Красноярск, ул. Республики, 51

# ИНФОРМАТИКА В ЗАДАЧАХ И УПРАЖНЕНИЯХ

Малое предприятие «Раско»